# การควบคุมการเคลื่อนที่ของเครน

# CRANE MOVEMENT CONTROL


โดย

นายนรพล ตั้งตรงจิตต์

นางสาวปัญจรักษ์ นิจสุนกิจ

นายภาคภูมิ สุขแสนถาวร


อาจารย์ที่ปรึกษา

รองศาสตราจารย์ ดร. จงกล งามวิวิทย์

อาจารย์ สุมิตร พนาอุดมทรัพย์



ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมระบบควบคุม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

ปริญญานิพนธ์ปีการศึกษา 2539

ภาควิชาวิศวกรรมระบบควบคุม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การควบคุมการเคลื่อนที่ของเครน

CRANE MOVEMENT CONTROL

ผู้จัดทำ นายนรพล ตั้งตรงจิตต์

นางสาวปัญจรักษ์ นิจสุนกิจ

นายภาคภูมิ สุขแสนถาวร

..... อาจารย์ที่ปรึกษา

(รองศาสตราจารย์ ดร. จงกล งามวิวิทย์)

..... อาจารย์ที่ปรึกษา

(อาจารย์ สุมิตร พนาอุดมทรัพย์)

# การควบคุมการเคลื่อนที่ของเครน

# CRANE MOVEMENT CONTROL

โดย

นาย นรพล ตั้งตรงจิตต์

รหัสประจำตัวนักศึกษา 36014210

นส.ปัญจรักษ์ นิจสุนกิจ

รหัสประจำตัวนักศึกษา 36014259

นายภาคภูมิ สุขแสนถาวร

รหัสประจำตัวนักศึกษา 36014316

อาจารย์ที่ปรึกษา

รศ.ดร.จงกล งามวิวิทย์

อาจารย์ สุมิตร พนาอุดมทรัพย์

บทคัดย่อ

โครงงานการควบคุมการเคลื่อนที่ของเครนในปีนี้ มีจุดมุ่งหมายที่จะพัฒนาโครงงานที่รุ่นพี่ได้ทำเอาไว้ให้ดียิ่งขึ้น โดยเพิ่มส่วนการควบคุมภาระให้สามารถควบคุมการเคลื่อนที่ขึ้นลงของตำแหน่งของภาระได้ตามต้องการ เพื่อประโยชน์ในการยกของผ่านสิ่งกีดขวาง โดยที่ยังสามารถรักษาสภาพการเคลื่อนที่ของตัวรถเครนให้ไปหยุด ณ ตำแหน่งที่ต้องการ พร้อมทั้งให้ภาระที่แขวนอยู่แกว่งน้อยที่สุด การออกแบบระบบควบคุมเพื่อควบคุมระบบเครนนี้ ได้ออกแบบโดยวิธี LQR

ABSTRACT

Crane Movement Control Project in this year has main purpose on improving the last year project by increasing the load control part. This load control part has the benefit that can avoid the undesired object between the trajectory of load. Increasing this part we can control the position of cart, the anti swing of load and the movement of rope the in the same time. This controller which has an ability to control this crane system is designed by LQR method.

## กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยดีก็เพราะได้รับความเมตตาจาก รองศาสตราจารย์ ดร. จงกล งามวิวิทย์ อาจารย์ สุมิตร พนาอุดมทรัพย์ อาจารย์ที่ปรึกษา รองศาสตราจารย์ ดร. โยธิน เปรมปราณีรัชต์ ดร. สุธี ผู้เจริญชนะชัย รองศาสตราจารย์ วิพันธ์ ปรีชาพานิช ดร. นนทวัฒน์ จุลเดชะ ดร. วันชัย ริ้วรุจา อาจารย์เกียรติศักดิ์ คมวัชระ อาจารย์เกียรติวรรณ ทรงสัตย์ และขอขอบคุณพี่สุณินูณ พี่นิรุตต์ พี่สุวัฒ พี่บุญเลิศ พี่เทพจิตร และพี่ ๆ ป.โท ที่ให้ความกรุณาแนะนำแก่ผู้จัดทำตลอดมา ผู้จัดทำขอกราบขอบพระคุณเป็นอย่างสูงมา ณ โอกาสนี้ด้วย และยังขอขอบคุณคณาอาจารย์ต่างๆ ที่ได้ประสิทธิประสาทวิชาความรู้แก่ผู้จัดทำ

ขอขอบคุณ บริษัท อีซูซุ(ประเทศไทย) จำกัด ที่ได้ให้ความอนุเคราะห์อุปกรณ์ที่ใช้ในระบบควบคุมการเคลื่อนที่ของเครน ซึ่งทำให้ปริญญานิพนธ์เรื่องนี้สำเร็จลงได้ด้วยดี

ขอกราบขอบพระคุณคุณพ่อคุณแม่ของผู้จัดทำที่ได้อุปการะผู้จัดทำและยังเป็นผู้ให้กำลังใจแก่ผู้จัดทำตลอดมาและต้องขอขอบคุณเพื่อน ๆ ที่ให้ความเป็นเพื่อนและให้กำลังใจแก่ผู้จัดทำตลอด

ผู้จัดทำ
นายนรพล ตั้งตรงจิตต์
นางสาวปัญจรักษ์ นิจสุนกิจ
นายภาคภูมิ สุขแสนถาวร

# สารบัญ

# บทที่1
# บทนำ

## 1.1 จุดประสงค์ของการศึกษาและการควบคุมการเคลื่อนที่ของเครน

จุดประสงค์หลักในการควบคุมการเคลื่อนที่ของเครนมีเหตุผลดังกล่าวต่อไปนี้

1.1.1 ช่วยเพิ่มประสิทธิภาพในการทำงานโดยเฉลี่ย แม้ว่าผู้ปฏิบัติงานจะไม่มีความถนัด

1.1.2 ช่วยเพิ่มประสิทธิภาพการทำงานในสภาวะแวดล้อมที่ไม่เหมาะสมเช่นการที่ผู้ปฏิบัติงานที่ต้องทำงานในบริเวณที่มีอุณหภูมิสูง หรือ ในบริเวณที่มีทัศนวิศัยไม่ดี เป็นต้น

1.1.3 ช่วยลดค่าจ้างแรงงาน

1.1.4 ช่วยทำให้งานเสร็จเร็วยิ่งขึ้น

1.1.5 สามารถประยุกต์ใช้งานกับกระบวนการผลิตในโรงงานอุตสาหกรรมได้อย่างมีประสิทธิภาพ

## 1.2 ประวัติการศึกษาการควบคุมเครน

ในปี คศ.1979 Mita และ Kannai ได้พัฒนาวิธีการควบคุมการแกว่ง ของภาระโดยวิธีการควบคุมออพติมัล(Optimum Control) เพื่อให้ใช้เวลาในการควบคุมน้อยที่สุด

ในปี คศ. 1987 J.W.Auerning ปรับปรุงวิธีแก้ปัญหาการควบคุมให้ใช้เวลาน้อยที่สุด (Minimum time control) โดยการเปลี่ยนค่าความยาวลวดที่แขวนภาระขณะทำการควบคุม

ในปี คศ. 1993 J.Shirai ได้ทำการพัฒนาวิธีการคำนวณแบบใหม่เพื่อใช้ในการควบคุมการแกว่งของภาระที่แขวนอยู่บนเครนที่ใช้ในการขนตู้คอนเทนเนอร์ภายในท่าเรือ

## 1.3 เนื้อหาที่จะกล่าวถึงในปริญญานิพนธ์ฉบับนี้

ในปริญญานิพนธ์ฉบับนี้จะประกอบไปด้วยเนื้อหา ดังนี้

1.3.1 โครงสร้างเชิงกล และวงจรทางอิเล็กทรอนิกส์ที่ใช้ในการวัดค่าสัญญาณต่าง ๆที่ใช้ในแบบจำลองเครน

1.3.2 การ์ดที่ใช้ติดต่อกับคอมพิวเตอร์ วงจรขับมอเตอร์ และวงจรรีเลย์

1.3.3 การประมาณค่าโมเดลทางคณิตศาสตร์ของระบบเครน

1.3.4 การควบคุมการเคลื่อนที่ของเครนโดยการป้อนกลับสเตท(state feedback)ซึ่งออกแบบค่าเกนป้อนกลับ(feedback gain)โดยวิธี Linear Quardratic Regulator(LQR) โดยจะมีค่าเกนเปลี่ยนไปตามค่าความยาวเชือก

1.3.5 การทดลองและผลการทดลอง

# บทที่ 2
## โครงสร้างโดยรวมของระบบ

### 2.1 หลักการในการควบคุมและบล็อกไดอะแกรมของระบบ

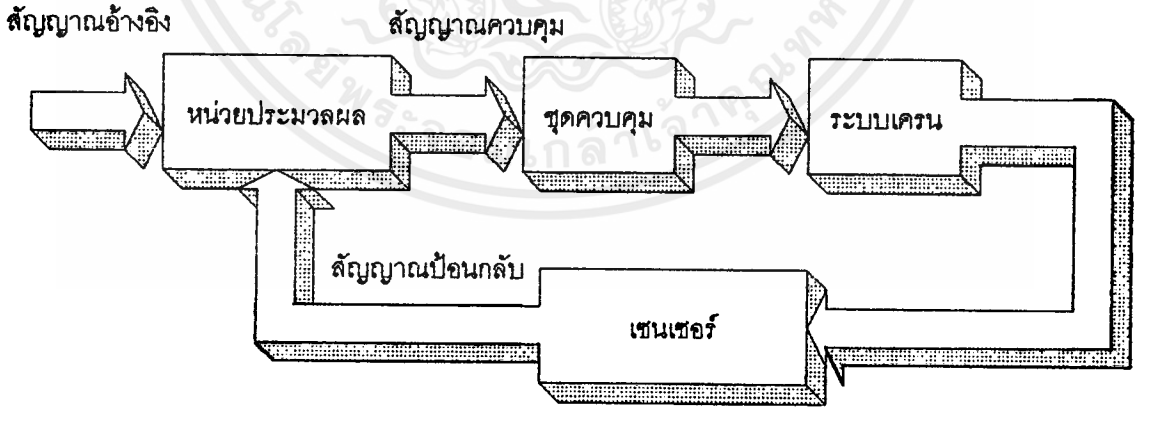
2.1.1 หลักการในการควบคุม

หลักในการควบคุมระบบการเคลื่อนที่ของเครนไปยังเป้าหมายพร้อมทั้งสามารถยกสิ่งกีดขวางและนำลงสู่เป้าหมายได้อย่างถูกต้องไม่แกว่งนั้น อาศัยแรงขับเคลื่อนจากมอเตอร์ เพราะฉะนั้นจำเป็นที่เราจะควบคุมการหมุนของมอเตอร์ให้ได้เพื่อที่จะควบคุมการเคลื่อนที่ของเครนและควบคุมมอเตอร์ในการดึงภาระหลบสิ่งกีดขวางวางลงสู่เป้าหมายและเกิดมุมในการแกว่งน้อยที่สุด

การควบคุมระบบเครนโดยควบคุมการหมุนของมอเตอร์ ได้จากการทำงานร่วมกันของคอมพิวเตอร์ ชุดควบคุม และเซนเซอร์ โดยเซนเซอร์จะทำการวัดสัญญาณป้อนกลับ (Feedback signal) ค่าตำแหน่งเครน ค่ามุมของการแกว่งและค่าความยาวเชือกที่ใช้ดึงภาระหลบสิ่งกีดขวาง มาคำนวณหาค่าสเตทต่าง ๆและสัญญาณควบคุม ไปยังชุดควบคุมเพื่อควบคุมให้เครนเคลื่อนที่ไปยังตำแหน่งที่ต้องการและมีมุมของภาระเป็นศูนย์ ไม่มีค่าผิดพลาดหรือมีแต่น้อย

2.1.2 บล็อกไดอะแกรมของระบบ

สามารถเขียนบล็อกไดอะแกรมโดยรวมของระบบได้ดังนี้

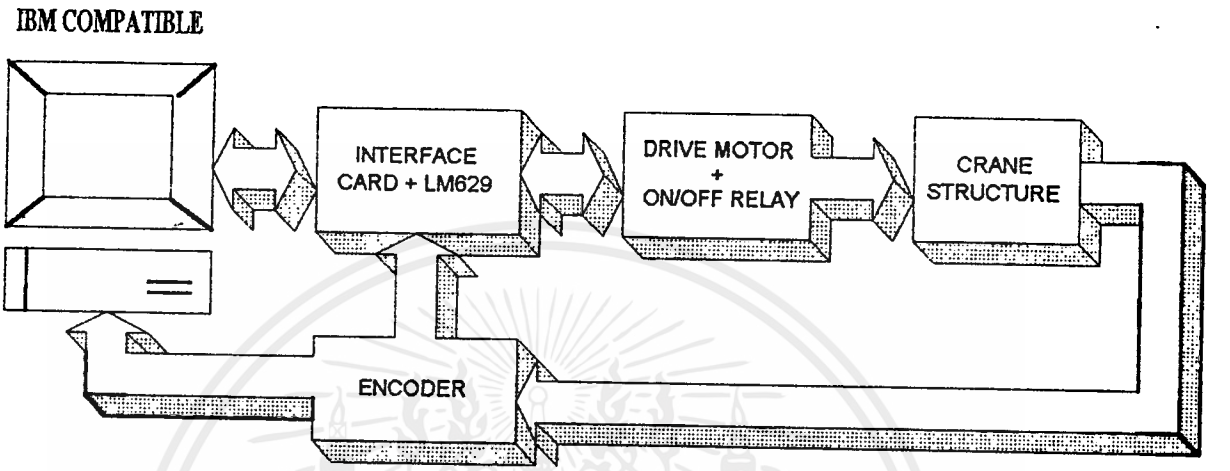



รูปที่2.1 รูปแสดงบล็อกไดอะแกรมโดยรวมของระบบการควบคุมการเคลื่อนที่ของเครน

2 .2 ชุดควบคุมการเคลื่อนที่และดึงภาระขึ้นลงของเครน

2 .2 .1ชุดควบคุมการเคลื่อนที่และยกภาระขึ้นลงของเครน

ประกอบด้วย




รูปที่2.2 รูปแสดงบล็อกไดอะแกรมของชุดควบคุมการทำงานของระบบเครน

2 .2.1.1คอมพิวเตอร์ส่วนบุคคล

ใช้ควบคุมระบบโดย software ที่ถูกเขียนขึ้นโดยทฤษฎีควบคุม โดยเลือกใช้คอมพิวเตอร์ที่ใช้กันได้กับของบริษัท IBM เพราะหาง่ายและใช้กันอย่างแพร่หลาย

2.2.1.2วงจรเชื่อมต่อระหว่างคอมพิวเตอร์ที่รวมชิพควบคุมความเร็ว

ลักษณะของวงจรเชื่อมต่อระหว่างคอมพิวเตอร์

เป็นวงจรที่ทำหน้าที่ช่วยให้คอมพิวเตอร์สามารถติดต่อส่งผ่านข้อมูลชิพควบคุมความเร็วกับวงจรขับมอเตอร์ วงจรดึงภาระขึ้นลงหรืออุปกรณ์ภายนอกได้

ในวงจร Interface จะมีส่วนที่ทำหน้าที่เป็นบัฟเฟอร์ของสัญญาณข้อมูล (data buffer) ซึ่งจากรูปที่ 2.7

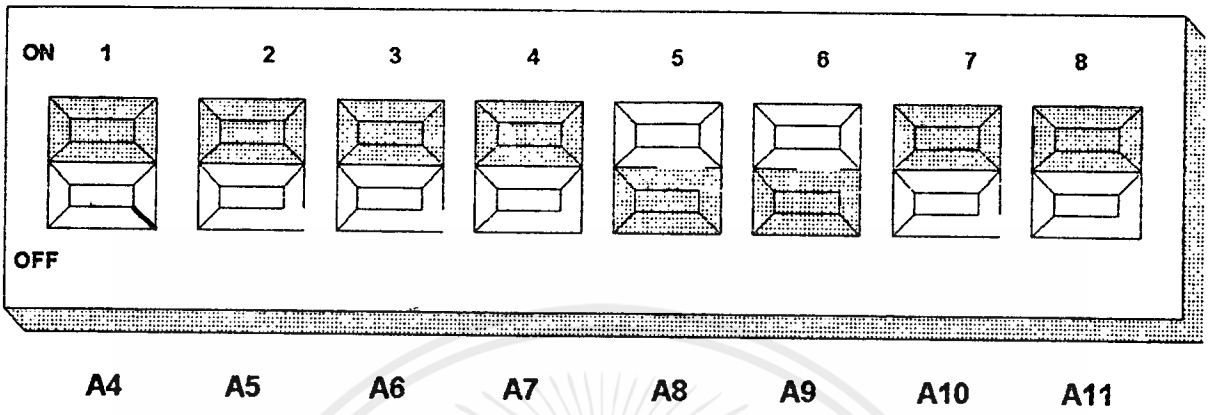
ใช้ IC 74LS245 ทำหน้าที่เป็นบัฟเฟอร์ทั้งรับและส่งผ่านข้อมูลใน 2 ทิศทาง

ใช้ IC 74LS244 เป็นบัฟเฟอร์ของสัญญาณควบคุม (signal buffer) ทำหน้าที่ส่งสัญญาณไปควบคุม การทำงานของ พอร์ทต่างๆในวงจรเชื่อมต่อได้ถูกต้องและผ่านแอดเดรสที่ต้องการ

โดยมีหลักในการกำหนดแอดเดรสดังนี้

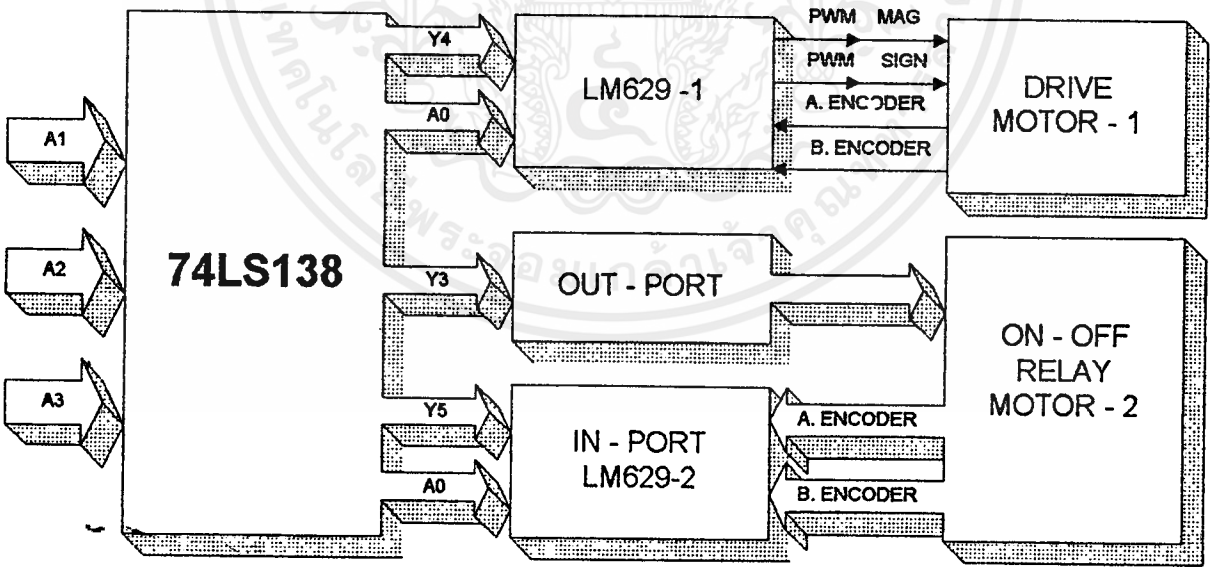
การกำหนดแอดเดรสเพื่อติดต่อกับพอร์ทต่างๆ ที่ใช้ควบคุมความเร็วจากการปิดเปิดของ ดิปสวิตช์ ของ IC 74LS688 ในการดีโคดแอดเดรสโดยเลือกแอดเดรสที่ 300H เท่ากับ 0011 0000 XXXX

โดยที่ A11=0 ,A10=0 ,A9=1 ,A8=1 ,A7=A6=A5=A4=0 ตามรูปที่ 2.3



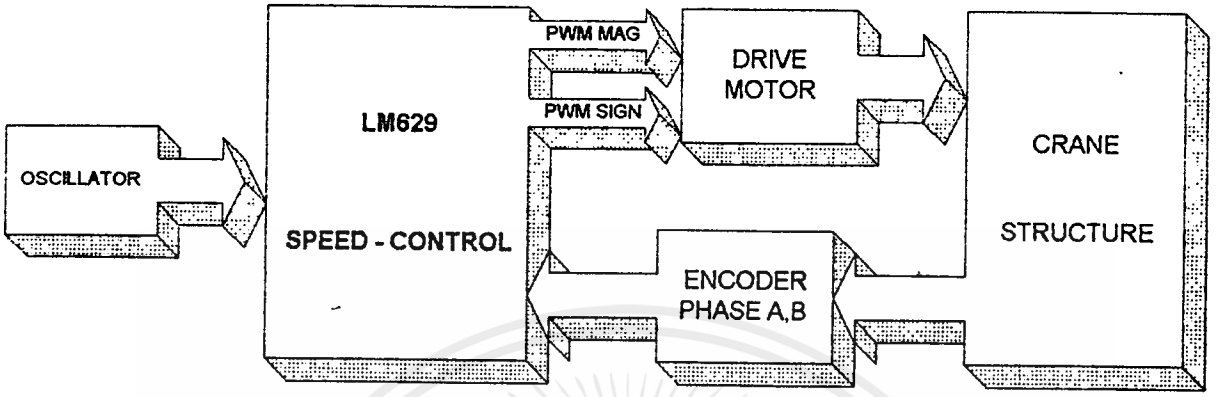

รูปที่2.3 แสดงการปิดเปิดของดิปสวิตส์

ส่วน A3,A2,A1และ A0 ถูกดีโคดด้วย IC 74LS138 อีกครั้งเพื่อกำหนดพอร์ทของชุดควบคุมความเร็วที่ติดต่อ ชิพ LM629 ตัวที่1 กับวงจรขับมอเตอร์ เพื่อกำหนด เอาท์พอร์ทของวงจรดึงภาระขึ้นลงและเพื่อกำหนด พอร์ทของการ อินพอร์ทจากเอนโคดเดอร์ของวงจร ดึงภาระขึ้นลง




รูปที่2.4 บล็อกไดอะแกรมแสดงการดีโคดแอดเดรสของ 74LS138 กับพอร์ทการใช้งานต่างๆ

ชิพควบคุมความเร็ว LM629




รูปที่2.5 บล็อกไดอะแกรมการทำงานชิพควบคุมความเร็ว LM629

IC LM629 เป็นชิพควบคุมการเคลื่อนที่ (Motion Conrol IC) ทำงานร่วมกับหน่วยประมวลผลหลัก ออกแบบมาให้ใช้ได้กับทั้ง ดีซีเซอร์โวมอเตอร์ และระบบเซอร์โวแมคคานิซมอื่นๆ มีการป้อนกลับของ เฟส A , B เอนโคดเดอร์นำมาคำนวณหาสัญญาณควบคุมไปควบคุมการเคลื่อนที่ของตัวเครน

คุณสมบัติของ LM629

- มีรีจีสเตอร์ตำแหน่งความเร็ว ความเร่งขนาด 32 บิต
- เป็นฟิลเตอร์ดิจิตอลแบบ PID
- สามารถกำหนดเส้นทางของความเร็ว (tarjectory)
- สามารถกำหนดค่าคาบการชักตัวอย่างได้
- ให้เอาท์พุตเป็น PWM ซึ่งมีทั้งขนาดและทิศทาง
- ค่าความเร็ว ค่าตำแหน่งที่ต้องการ และค่าพารามิเตอร์ของฟิลเตอร์สามารถเปลี่ยนแปลงได้ขณะที่ทำการควบคุมจริง
-สามารถทำงานได้ทั้งในโหมดการควบคุมตำแหน่งและโหมดการควบคุมความเร็ว
- สามารถโปรแกรม การทำงานขัดจังหวะ(interrupt) กับหน่วยประมวลผลหลักได้
- ติดต่อข้อมูลกับหน่วยประมวลผลหลักแบบขนาน ขนาด 8 บิตได้
- มีหน่วยรับสัญญาณการป้อนกลับจากเอนโคดเดอร์

การติดต่อ LM629 หนึ่งตัวนั้นมี 2 แบบดังนี้

แบบที่ 1 เป็นการติดต่อเพื่อส่งสัญญาณคำสั่งงานให้ LM629 เพื่ออ่านค่าแฟล็กแสดงสถานะของ LM629 ในการติดต่อแบบนี้ต้องให้สัญญาณที่ขา 12 และ 16 ของ LM629 เป็น 0

แบบที่ 2 เป็นการติดต่อเพื่อการอ่านข้อมูลและการเขียน ในการติดต่อแบบนี้ให้สัญญาณที่ขา 12 ของ 629 เป็น 0 และขาที่ 16 ของ LM629 เป็น 1

เนื่องจากการติดต่อกับ LM629 มีการติดต่อ 2 แบบเราจึงต้องใช้แอดเดรสในการติดต่อกับ LM629 2 แอดเดร้ส ต่อ 1 ตัว

2.2.1.3 วงจรขับมอเตอร์และวงจรดึงภาระขึ้นลงของเครน

การทำงานของวงจรขับมอเตอร์

เมื่อป้อน แรงดันไฟตรงให้วงจรมากกว่า 12 โวล์ท แต่ไม่เกิน 40 โวล์ท พร้อมทั้งได้สัญญาณ PWM MAG และ PWM SIGN จากชิพ LM629 ตัวที่1 บนอินเทอร์เฟสการ์ด

โดย ค่า PWM MAG เป็น 0 มอเตอร์จะหยุดทำงาน

ค่า PWM MAG เป็น 1 มอเตอร์จะทำงานตามสัญญาณ PWM SIGN

ค่า PWM SIGN เป็น 1 มอเตอร์จะหมุนตามเข็มนาฬิกา

ค่า PWM SIGN เป็น 0 มอเตอร์จะหมุนทวนเข็มนาฬิกา

ในขณะที่เครนเคลื่อนที่เอนโคดเดอร์ทำหน้าที่เป็นเซนเซอร์ส่งผ่าน เฟส A,B ไปยังชิพควบคุมความเร็ว LM629 ตัวที่ 1 เพื่อเป็นเคาท์เตอร์คำนวณระยะทางที่เครนเคลื่อนที่ไปได้และให้เครนหยุดได้ตามต้องการ

การทำงานของวงจรดึงภาระขึ้นลง

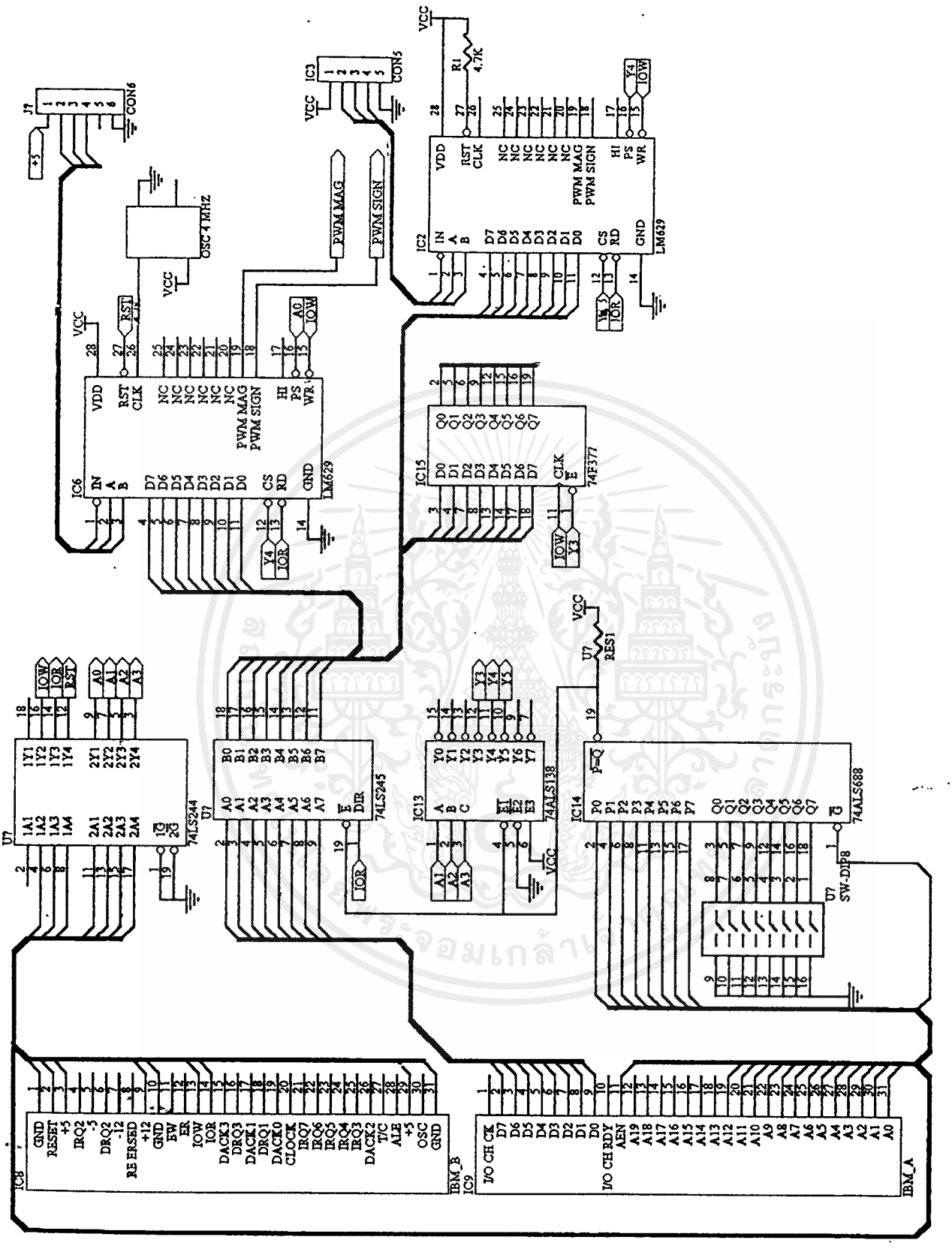
จากวงจรเราใช้รีเลย์ 2 ตัวในการควบคุมการทำงานของมอเตอร์ที่ใช้ดึงภาระของเครนขึ้น หยุดมอเตอร์ และปล่อยภาระของเครนลงโดย

โดยใช้รีเลย์แบบ 2 หน้าสัมผัส ในการเปลี่ยนทิศทางของมอเตอร์

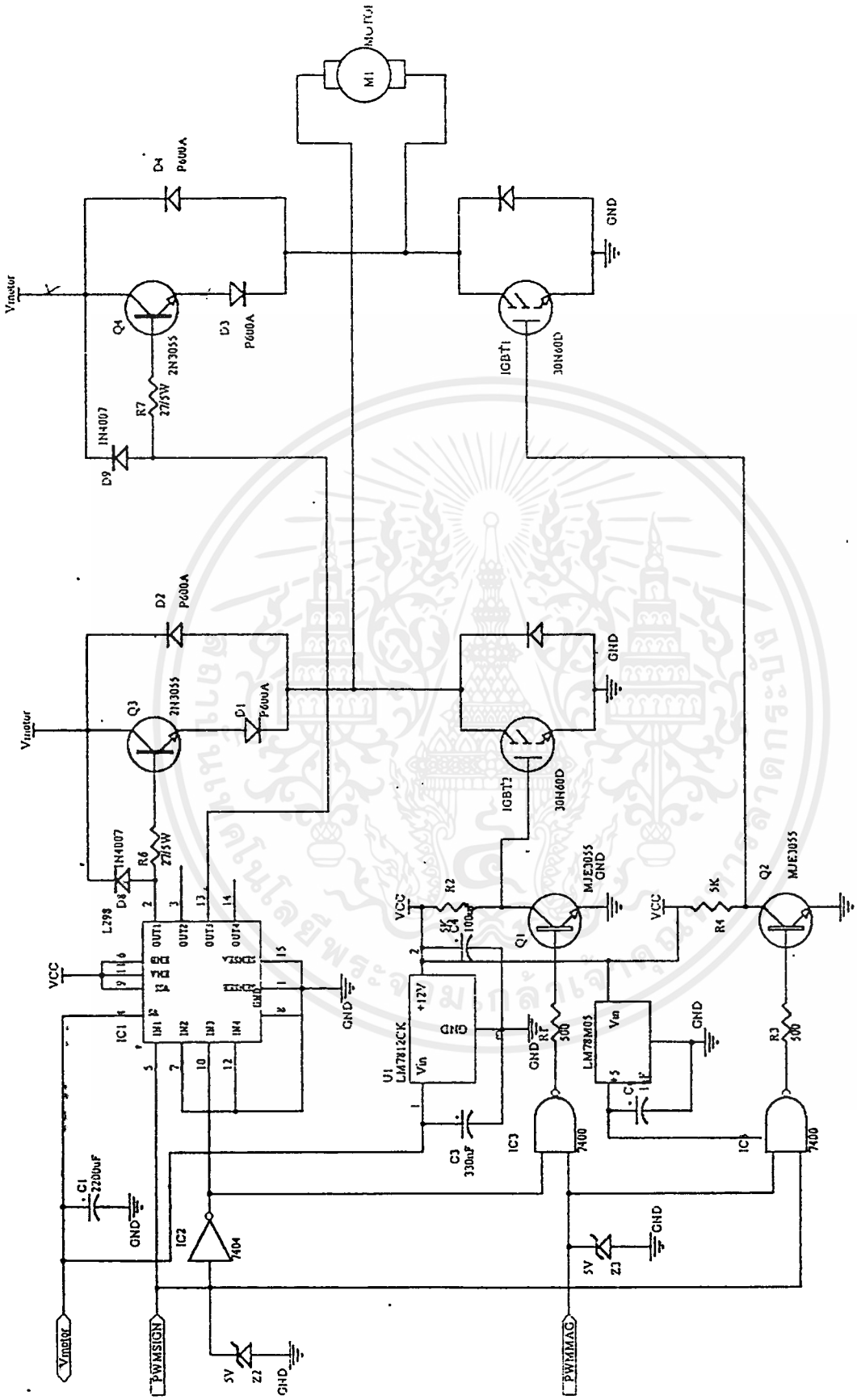
โดยใช้รีเลย์แบบ 1 หน้าสัมผัส ในการควบคุมให้มอเตอร์ทำงานหรือหยุดทำงาน

ซึ่งเราได้สัญญาณควบคุมการทำงานของรีเลย์ทั้ง 2 ตัวผ่าน เอาท์พอร์ทของ 74LS377 บนบอร์ดอินเทอร์เฟส โดยเมื่อสัญญาณ OD1 เป็นศูนย์ จะทำให้มอเตอร์หยุดทำงาน แต่เมื่อ OD1 เป็นหนึ่งเครนจะทำงานดึงภาระขึ้นก็ต่อเมื่อ OD0 เป็น 1 ปล่อยภาระลงเมื่อ OD0 เป็น 0

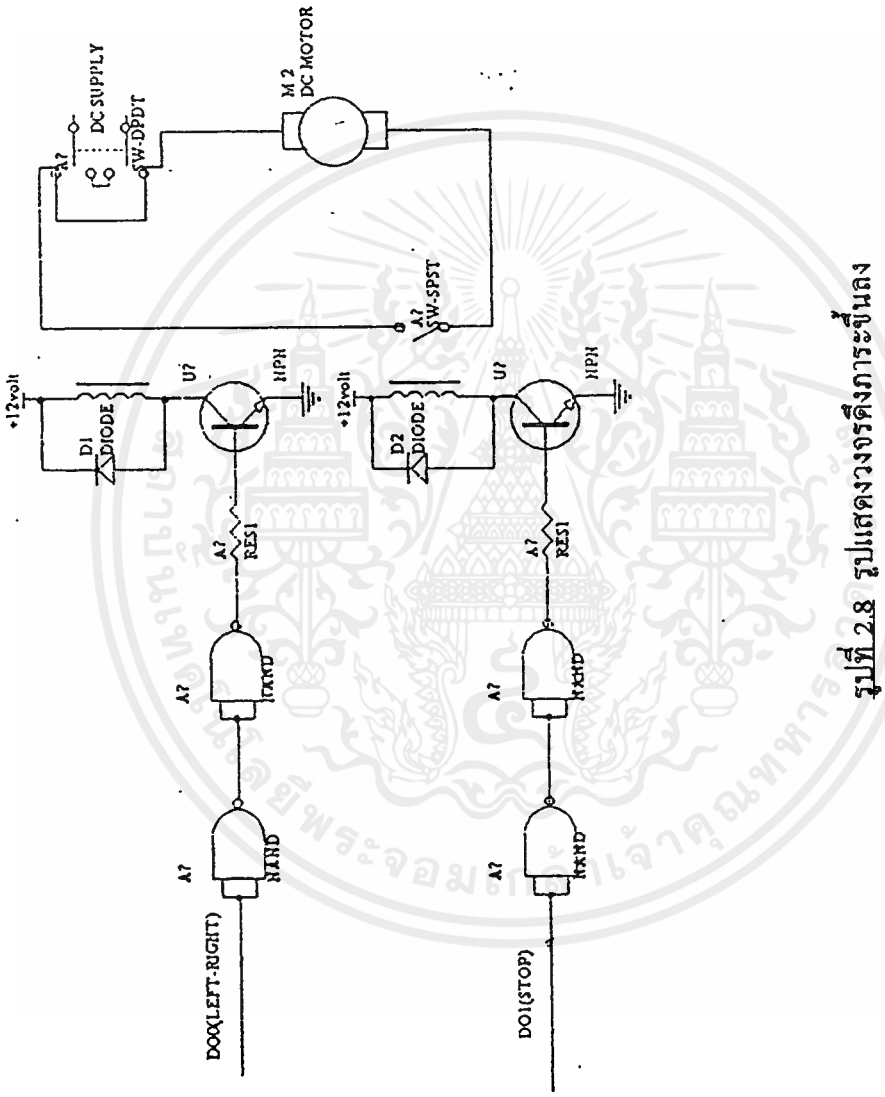
ในขณะที่เครนดึงภาระขึ้นหรือปล่อยภาระลงเอนโคดเดอร์จะถูกส่งค่าเฟส A,B ส่งผ่านกลับไปให้ ชิพ LM629 ตัวที่ 2 ซึ่งใช้เป็น เคาท์เตอร์นับความยาวเชือกที่ได้ส่งไปให้ คอมพิวเตอร์ประมวลผลส่งสัญญาณไปควบคุมการทำงานผ่าน เอาท์พอร์ท 74LS377 OD0,OD1 ไปควบคุมการหมุนของมอเตอร์ที่ทำหน้าที่ดึงภาระขึ้นลง




รูปที่ 2.6 รูปแสดงวงจรเชื่อมต่อระหว่างคอมพิวเตอร์กับวงจรควบคุมความเร็ว LM629




รูปที่ 2.7 รูปแสดงวงจรขับมอเตอร์




รูปที่ 2.8 รูปแสดงวงจรดึงภาระขึ้นลง

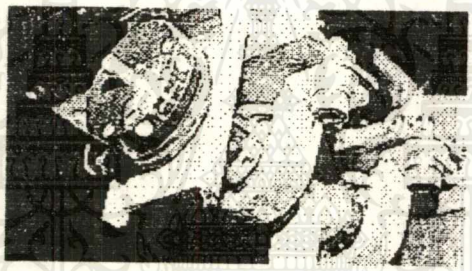
2.3 การเลือกใช้เซ็นเซอร์และการออกแบบวงจรเซ็นเซอร์

เซ็นเซอร์เป็นส่วนประกอบที่สำคัญในระบบคอนโทรลที่มีการป้อนกลับเพราะสัญญาณที่ป้อนกลับสามารถนำมาวิเคราะห์หาคุณสมบัติของระบบหรือสามารถนำมาหาสถานะของระบบหรือสามารถบอกสถานะของระบบขณะนั้นได้ว่าเป็นอย่างไรซึ่งจะได้นำมาหาสัญญาณควบคุมที่จะส่งไปควบคุมระบบ ดังนั้นถ้าสัญญาณป้อนกลับที่ได้ถูกต้องมากและมีสัญญาณรบกวนน้อยก็จะทำให้เราได้ค่าสัญญาณควบคุมที่สามารถควบคุมระบบให้เกิดประสิทธิภาพสูงสุดได้

ในระบบเครนใช้เซ็นเซอร์ 2 ชนิด

ก โพเทนทิโอมิเตอร์ ( Potentiometer)

ใช้วัดค่ามุมของการแกว่งของภาระ



รูปแสดงโพเทนทิโอมิเตอร์

Potentiometer เป็น R ปรับค่าได้ชนิดหนึ่งซึ่งมีแกนกลางหมุนทำให้ค่า R เปลี่ยนแปลง มี 2 ชนิดที่นิยมใช้ คือ

- ชนิด Plastic flim จะมีความละเอียดของค่าสูง ( ขนาดสูงสุด 2 กิโลโอห์ม)

-ชนิด Wire wound ความละเอียดจะต่ำกว่าแบบ Plastic film แต่สามารถทนกระแสได้ดีกว่า

ค่าความต้านทานที่เกิดขึ้นจะมีความสัมพันธ์กับความต้านทานเชิงมุมเป็นเชิงเส้น

เราได้ทำการคิดโพเทนทิโอมิเตอร์ไว้ด้านล่างของตัวรถเครนและยึดแกนการหมุนของโพเทนทิโอมิเตอร์กับแกนการแกว่งของภาระของเครน ทำให้เมื่อมีการแกว่งของภาระก็มีผลทำให้ค่าความต้านทานของโพเทนทิโอมิเตอร์เปลี่ยนไปด้วย ซึ่งค่าความต้านทานของโพเทนทิโอมิเตอร์จะมีความสัมพันธ์กับมุมของการแกว่งเป็นเชิงเส้น

วงจรตรวจวัดค่ามุมโดยใช้โพเทนทิโอมิเตอร์

เนื่องจากค่าความสัมพันธ์ของความต้านทานโพเทนทิโอมิเตอร์กับค่ามุมในการแกว่งเป็นเชิงเส้น ดังนั้นเราจึงออกแบบวงจรที่ใช้ตรวจวัดค่ามุม โดยใช้โพเทนทิโอมิเตอร์มีหน้าที่เปลี่ยนค่ามุมที่ได้มาเป็นโวลท์เตจ โดยใช้การจ่ายแรงดันคงที่ให้กับโพเทนทิโอมิเตอร์ที่ได้จะถูกนำมาเปรียบเทียบกับค่าแรงดันที่ได้

จากค่าความต้านทานปรับค่าได้ R6 (จากรูปที่ 2.6 ) ซึ่งใช้เป็นแรงดันอ้างอิงและแรงดันเปรียบเทียบจะถูกขยายสัญญาณโดย Instrument Amplifier ซึ่งอัตราขยายเป็นแบบ differential ที่สามารถปรับค่าอัตราขยายได้จากค่าความต้านทาน R4 และมีความสัมพันธ์ระหว่างค่าแรงดันเอาท์พุตและแรงดันอินพุตดังนี้

$$Vout = (\frac{R_{10}}{R_8 R_a})(R_a + R_5 + R_3)(V_{pt} - V_{ref})$$

เมื่อ $R_a = R_4 + R_{4.1}$

$R_8 R_{11} = R_9 R_{10}$

$R_8 = R_{11} = R_9 = R_{10} = 10$ กิโลโอมห์

$R_5 = R_3 = 5$กิโลโอมห์

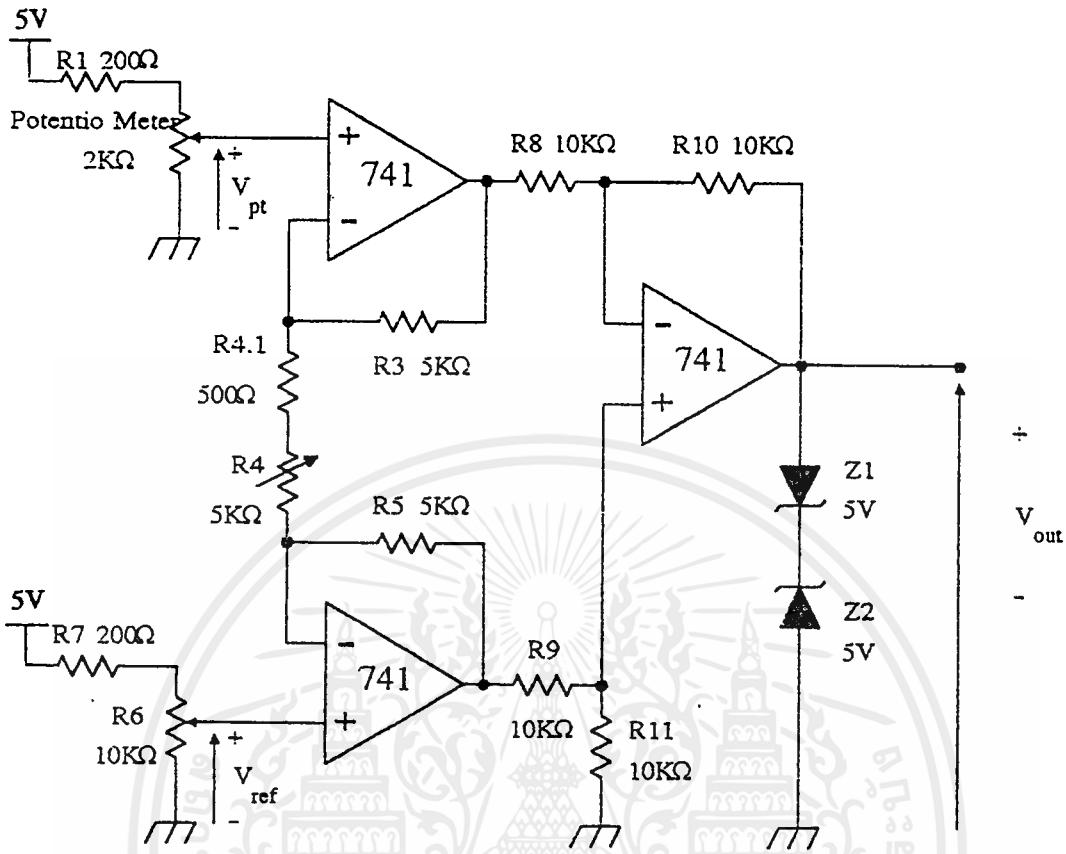
$V_{out}$ คือค่าแรงดันเอาท์พุตและจะมีค่าอยู่ในช่วง -5 V ถึง +5 V

$V_{pt}$ คือค่าแรงดันที่ได้จากการหมุนของโพเทนทำโอมิเตอร์

$V_{ref}$ คือค่าแรงดันอ้างอิงที่ได้จากการปรับค่าความต้านทาน $R_6$

จากความสัมพันธ์ในสมการจะเห็นว่า ค่าแรงดันเอาท์พุต ($V_{out}$) จะมีค่าความสัมพันธ์กับแรงดันที่ได้จากค่าความต้านทานของโพเทนทิโอมิเตอร์ ดังนั้นค่า $V_{out}$ ก็จะมีความสัมพันธ์กับค่ามุมของการแกว่งที่เกิดขึ้นเป็นเชิงเส้นเช่นกัน

หลังจากได้ค่ามุมป้อนกลับที่แปลงเป็นสัญญาณโวล์ทเตจจากวงจรเซ็นเซอร์แล้วเราก็จะนำสัญญาณที่ได้มาผ่าน Low-Pass Filter ที่ออกแบบและทำขึ้นบนบอร์ด ACLD-780 Screw Terminal Panel เพื่อกรองสัญญาณรบกวนที่ความถี่สูงออก และก่อนจะทำการแปลงสัญญาณจากอนาลอกเป็นดิจิตอลโดยผ่านการ์ด PCL-714 ที่ให้ความละเอียดของสัญญาณดิจิตอลถึง 14 บิตเพื่อทำให้สัญญาณที่ได้มาสามารถนำไปคำนวณในคอมพิวเตอร์ได้




รูปที่2.9 รูปแสดงวงจรตรวจวัดค่ามุมโดยใช้โพเทนทิโอมิเตอร์

PCL-714

PCL-714 (super - lab card) ประกอบด้วย วงจร A/D,D/A I/O card และยังมี Function ควบคุมสำหรับประยุกต์ใช้งานบน Board ตัวนี้ด้วย การ์ดตัวนี้สามารถนำไปประยุกต์ใช้งานได้หลายอย่าง มีความละเอียดสูงคือมีผลตอบสนองถึง 14 บิต

การ์ดตัวนี้ยังทำงานเป็นการ์ดหลักใช้งานร่วมกับการ์ดย่อยอื่นๆ เช่น PCLD-782, PCLD-782 และ PCLD-785 เป็นต้น ทำให้ การ์ดตัวนี้มีประสิทธิภาพสูงขึ้นไปอีก

ลักษณะของ PCL-714

- input ที่เป็น analog ทั้งหมด 16 สัญญาณ แต่ละสัญญาณมีความละเอียด 14 บิต
- output ที่เป็น analog 2 สัญญาณ ซึ่งแต่ละสัญญาณมีความละเอียดถึง 14 บิต
- A/D conversion ใช้เวลาอย่างน้อย 40 microsec
- มี input ที่เป็น digital 16 ค่า
- มี output ที่เป็น digital 16 ค่า
- ทั้ง input และ output channel ใช้ได้กับ TTL
- สามารถ program ช่วงเวลา timer ได้

- สามารถ program ช่วงเวลา trigger ได้

ตัวอย่างการนำ PCL-714 ไปใช้งาน

- ใช้เป็น digital input/output control

- ใช้ในการควบคุม process

- ใช้ในการวัดอุตสาหกรรม automation

- ใช้ในการวัดในห้องทดลอง automation

- ใช้เป็นวงจรนับความถี่

- ใช้เป็นวงจร A/D , D/A เป็นต้น

ข้อควรระวัง สำหรับการใช้การ์ด PCL-714

- เนื่องจาก PCL-714 ตัวนี้ใช้ 4052 cmos chip ในการ multiplex สำหรับ analog input channel ห้ามไม่ให้โวล์ตเตจ สูงกว่า 12.5 volt หรือต่ำกว่า -5.5 volt ซึ่งถ้า analog input channel ได้รับนอกเหนือจากช่วงนี้แล้ว จะทำให้วงจร multiplexer เสียหายได้

สำหรับ PCL-714 ตัวนี้ถูกนำมาใช้งานในส่วนของวงจร A/D คือเมื่อ load เกิดการแกว่ง Potentiometer จะรับค่าการเปลี่ยนแปลงซึ่งเป็นโวล์ตเตจ ที่เป็น analog มีค่าอยู่ในช่วง -5 volt ถึง 5 Volt PCL-714 ใช้วงจร A/D แปลงจากสัญญาณ analog เป็น digital ส่งไปให้คอมพิวเตอร์เพื่อทำการประมวลผลในส่วนของการควบคุมต่อไป (ดูรายละเอียดเพิ่มเติมได้ในภาคผนวก)

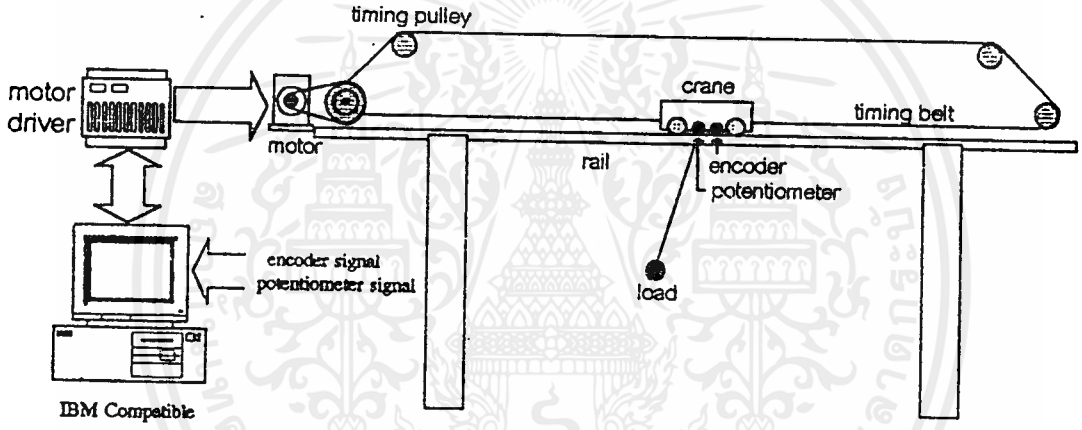
ข. เอนโคดเดอร์ (Encoder)

เอนโคดเดอร์ที่ใช้ในระบบมี 2 ตัว ตัวแรกใช้วัดระยะทางของตัวเครนติดตั้งไว้ที่ด้านข้างของตัวเครนเมื่อเครนเคลื่อนที่ไปก็จะทำให้ลูกยางที่ยึดติดกับแกนของเอนโคดเดอร์เคลื่อนที่ไปด้วย ตัวที่สองใช้วัดระยะทางของความยาวเชือกที่ใช้ดึงภาระขึ้นลงของเครน ติดตั้งไว้กับแกนของมอเตอร์ที่ทำหน้าที่ดึงภาระขึ้นลงลง เมื่อเครนเคลื่อนที่ มอเตอร์ตัวที่ทำหน้าที่ดึงภาระขึ้นลงจะทำงานเอนโคดเดอร์ที่ติดอยู่ก็จะทำงานตาม สัญญาณที่ได้จากเอนโคดเดอร์ถูกป้อนเข้าที่ ไอซี LM629 และเราสามารถอ่านค่าระยะทางที่เครนเคลื่อนที่พร้อมทั้งความยาวเชือกที่ใช้ดึงภาระขึ้นลง ได้จากคอมพิวเตอร์อ่านข้อมูลจากรีจีสเตอร์ของไอซี LM629

2.4 โครงสร้างและส่วนประกอบของแบบจำลองเครน

2.4.1 การออกแบบโครงสร้างทางกลของแบบจำลองเครน

แบบจำลองเครนที่ออกแบบไว้ใช้รางที่ทำด้วยสแตนเลสมีระยะทางวิ่งประมาณ 2 เมตร และสามารถยกภาระขึ้นได้สูงประมาณ 80 เซนติเมตร การวิ่งในแนวระนาบใช้ดีซีมิเตอร์เป็นตัวส่งกำลัง ส่งกำลังไปขับเคลื่อนรถเครนผ่านระบบส่งกำลังเพื่อให้ได้แรงบิดสูงขึ้น 3เท่า และให้มีค่าแบ็คแลช (back latch)และค่า สลิป(slip) น้อยที่สุด เพื่อให้ได้ความแม่นยำและเร็วที่สุดในการควบคุมการเคลื่อนที่ของตัวเครน ส่วนการดึงภาระขึ้นลง ได้ออกแบบให้ใช้ดีซีมอเตอร์เป็นต้นกำเนิดกำลัง เอนโคดเดอร์กำเนิดสัญญาณพัลส์ A,B เพื่อช่วยในการวัดระยะทาง และแกนกลางในการแกว่งเพื่อให้ภาระแกว่งได้ ดึงภาระขึ้นลงได้




รูปที่ 2.4ก แสดงโครงสร้างและส่วนประกอบของแบบจำลองเครน

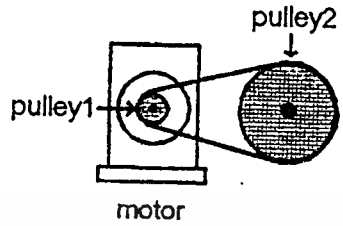
คุณสมบัติของส่วนประกอบต่างๆของแบบจำลองเครน

2.4.1.1 ตัวส่งกำลัง ในที่นี้เลือกใช้ดีซีมอเตอร์แบบแม่เหล็กถาวรเป็นตัวส่งกำลังไปยังตัวเครน ซึ่งให้อัตราส่วนระหว่างและแสอาร์เมเจอร์ และแรงบิดจะมีค่าคงที่ ให้ความสัมพันธ์ระหว่าง กระแสอาร์เมเจอร์ แรงบิดและความเร็ว เป็นเชิงเส้น จึงทำให้หาโมเดลได้ง่ายและไม่มีกำลังสูญเสียในฟิลด์ มีประสิทธิภาพสูงกว่าให้ค่ากระแสอาร์เมเจอร์สูงกว่าและขนาดเล็กกว่าเมื่อเทียบกับดีซีมอเตอร์แบบฟิลด์เป็นขอลวดที่มีขนาดแรงม้าเท่ากัน

2.4.1.2 ระบบส่งกำลัง เลือกใช้ระบบส่งกำลังที่ประกอบด้วย

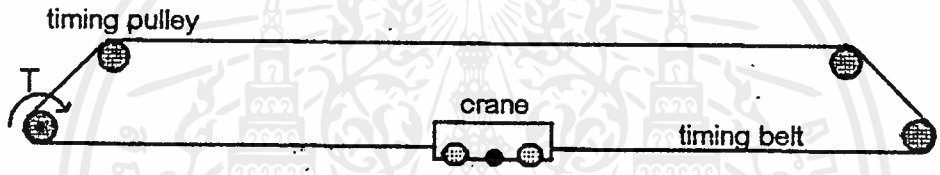
ก.ชุดทดรอบ ประกอบด้วยสายพานแบบไทม์มิ่งเบลท์(timing belt)และไทม์มิ่งพูลเลย์(timing pulley) โดยใช้ไทม์มิ่งพูลเลย์ 2 ขนาด ที่มีอัตราส่วนของรัศมีของไทม์มิ่งพูลเลย์ที่ยึดอยู่กับมอเตอร์เป็น 1/3 ของไทม์มิ่งพูลเลย์ที่ยึดดับแกนหมุน ของชุดส่งกำลังไปยังระบบ(ดังรูป2.4 ข)เมื่อทดรอบมอเตอร์ให้

ความเร็วลดลงแต่ให้แรงบิดเพิ่มขึ้น 3 เท่า การใช้ไทม์มิ่งเบลท์และไทม์มิ่งพูลเลย์นี้สามารถลดแบ็คแลชที่เกิดขึ้นในระบบทดรอบที่ใช้เฟืองเกียร์ได้




รูป 2.4ข. รูปแสดงการส่งกำลังในแบบที่มีการทดรอบของไทม์มิ่งพูลเลย์

2.4.1.3ชุดส่งกำลังประกอบด้วยสายพานแบบไทม์มิ่งเบลท์กับไทม์มิ่งพูลเลย์ ส่งกำลังไปยังตัวเครนโดยตรงดังรูป (2.4ค) เพื่อลดสลิปที่เกิดขึ้นในระบบที่ส่งกำลังโดยสลิงหรือสายพานธรรมดา



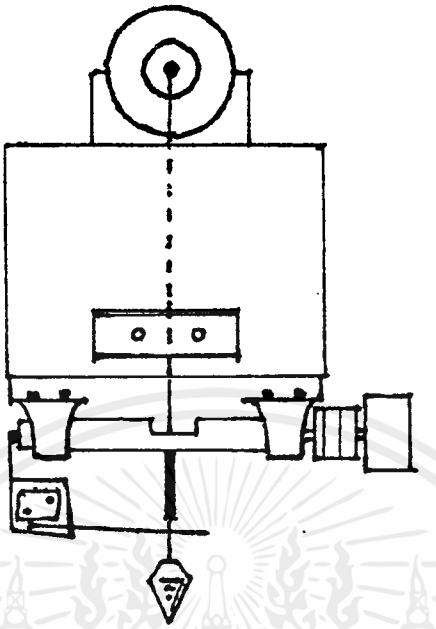

รูป 2.4ค รูปแสดงการส่งกำลังในแบบที่ไม่มีการทดรอบของไทม์มิ่งพูลเลย์

2.4.1.4ต้นกำเนิดกำลัง เราเลือกใช้ดีซีมอเตอร์แบบแม่เหล็กถาวรที่มีชุดทดรอบ 14 รอบ เป็นต้นกำเนิดกำลังในการดึงภาระขึ้นลง ทั้งนี้เพื่อที่จะสามารถกำหนดตำแหน่งของภาระให้อยู่ระหว่างจุดสูงสุดและต่ำสุดได้ โดยมีอัตราส่วนระหว่างกระแสอาร์เมเจอร์และแรงบิดจะมีค่าคงที่ ให้ความสัมพันธ์ ระหว่างกระแสอาร์เมเจอร์ แรงบิดและความเร็วเป็นเชิงเส้น

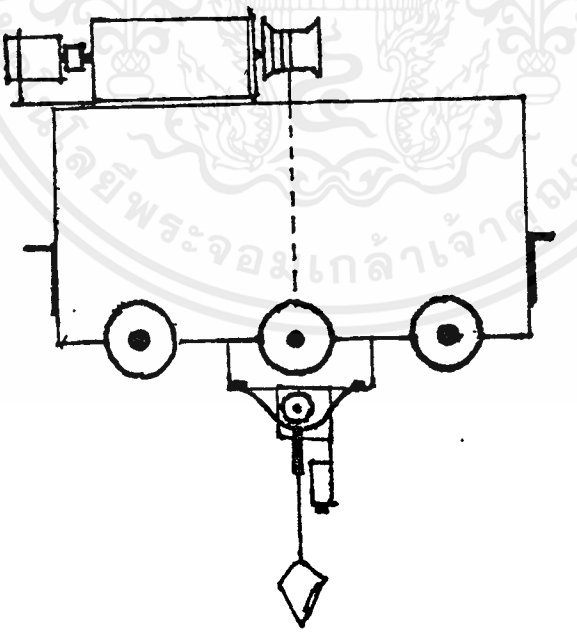
2.4.1.5เอนโคดเดอร์เป็นอุปกรณ์ที่ใช้สร้างพัลส์ที่แปรผันตรงกับการหมุนของมอเตอร์เพื่อใช้ในการวัดระยะทางและทิศทางการเคลื่อนที่ของภาระในแนวดิ่ง โดยติดตั้งเอนโคดเดอร์ไว้ที่ด้านบนของตัวเครนและเชื่อมต่อกับแกนที่ยื่นออกมาในส่วนท้ายของมอเตอร์เมื่อมอเตอร์หมุน เอนโคดเดอร์จะถูกป้อนให้กับไอซี LM629 ทำให้สามารถอ่านค่าระยะทางและทิศทางที่ภาระเคลื่อนที่ได้โดยใช้คอมพิวเตอร์อ่านข้อมูลจากรีจีสเตอร์ของไอซี LM629(ดูรายละเอียดในภาคผนวก)

เอนโคดเดอร์ที่ใช้มีทั้งหมด 5ขา คือ ขากราวนด์ ขาไฟเลี้ยง 5 โวลท์ ขาเอาท์พุทเฟส Aขาเอาท์พุทเฟส B และขา Index กำเนิดพัลส์ได้ 500 พัลส์ ต่อรอบ

2.4.1.6 limit swith เป็นอุปกรณ์ ตัดต่อวงจร ติดตั้งอยู่ที่แกนของการแกว่งของภาระมีไว้เพื่อป้องกันไม่ให้ภาระวิ่งชนโครงสร้างส่วนอื่น โดยเป็นชนิดปกติปิด และต่ออนุกรมเข้าที่ขั้วด้านใดด้านหนึ่งของมอเตอร์ ฉะนั้นถ้าภาระวิ่งชนก้านของ limit switch ก็จะทำให้ limit switch เปิดวงจรมอเตอร์ก็จะหยุดหมุน



รูปที่ 2.4ง รูปแสดงภาพด้านหน้ารถเครน

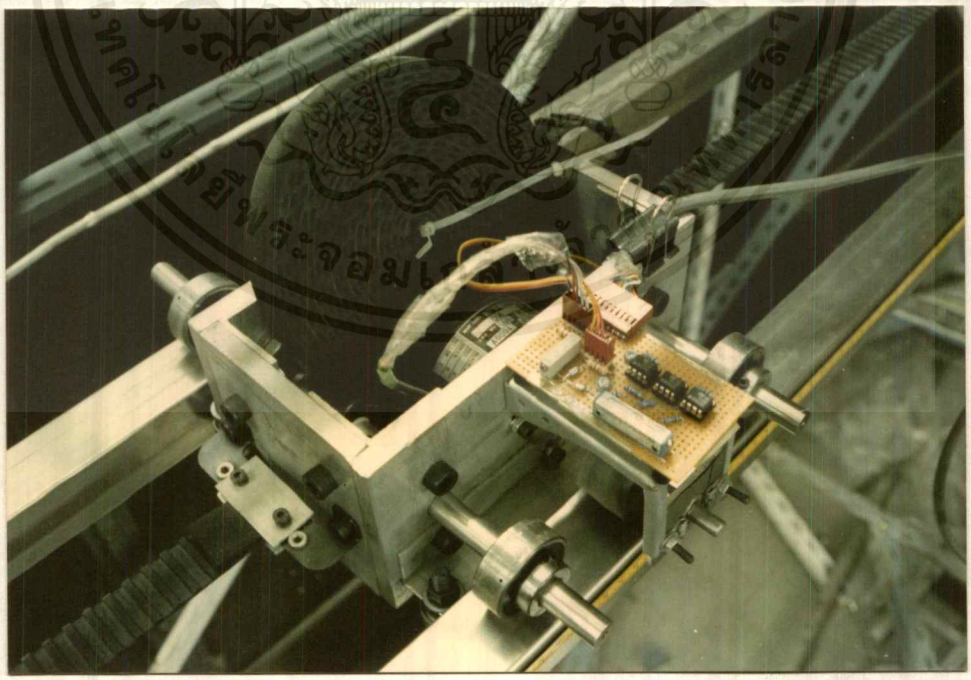


รูปที่ 2.4จ รูปแสดงภาพด้านข้างรถเครน

รูปที่2.4ฉ รูปแสดงระบบโดยรวมของเครน

รูปที่2.4ช รูปแสดงโครงสร้างทางกลทั้งหมดของระบบเครน

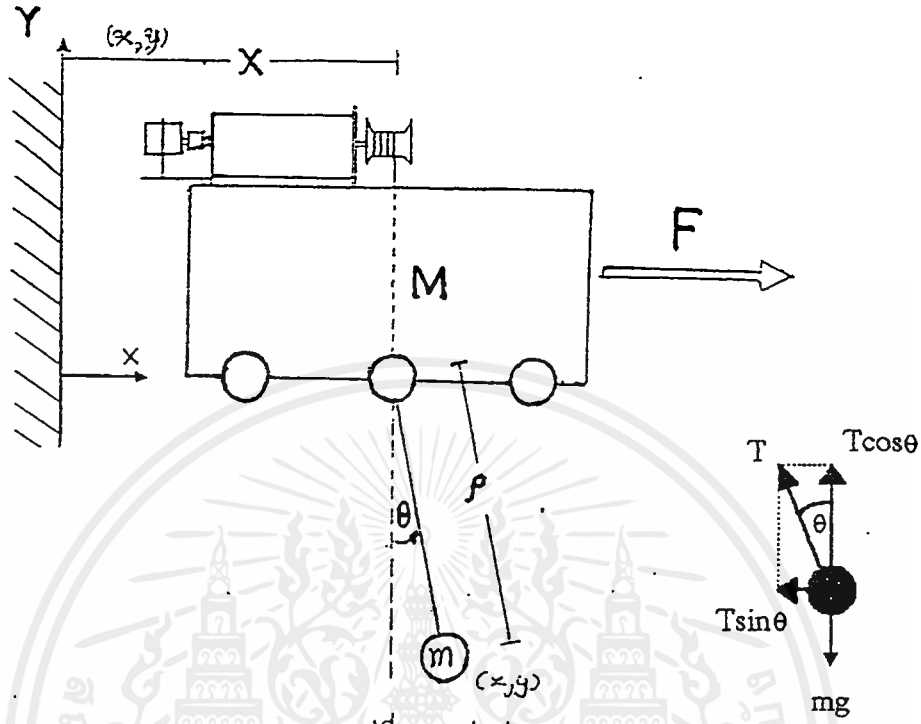
รูปที่ 2.4ช รูปวงจรขับมอเตอร์และดึงภาระขึ้นลงของเครน



รูปที่ 2.4ฌ รูปแสดงเอนโคดเดอร์และมอเตอร์ที่ใช้ดึงภาระขึ้นลง

<u>รูปที่ 2.4ญ</u> รูปแสดงภาระของเครน

## 2.5 โมเดลทางคณิตศาสตร์ของแบบจำลองเครน




รูปที่2.5 รูปแสดงแบบจำลองเครน

โดย

M: มวลของตัวเครน (kg)

$\chi$: ระยะทางของเครนที่เคลื่อนไปได้ตามแกนนอน เมื่อเทียบกับจุดอ้างอิง (m)

$\gamma$: ระยะทางของเครนที่เคลื่อนไปได้ตามแกนตั้ง เมื่อเทียบกับจุดอ้างอิง (m)

m: มวลของภาระ (kg)

x: ระยะทางของภาระที่เคลื่อนที่ไปได้ตามแกนนอน เมื่อเทียบกับจุดอ้างอิง (m)

y: ระยะทางของภาระที่เคลื่อนที่ไปได้ตามแกนตั้ง เมื่อเทียบกับจุดอ้างอิง (m)

$\rho$: ความยาวของเชือกที่แขวนภาระ (m)

g: ความเร่งโน้มถ่วงของโลก (m/s2)

F: แรงที่กระทำกับตัวเครน (N)

รูปที่2.5

ความสัมพันธ์ $x = \chi + \rho\sin\theta$ (1)

$y = -\rho\cos\theta$ (2)

รูปที่2.5 T = แรงตึงในเส้นเชือก

$m\ddot{x} = -T\sin\theta$ (3)

$m\ddot{y} = T\cos\theta - mg$ (4)

จาก (3),(4) ได้

$$\ddot{x}\cos\theta + \ddot{y}\sin\theta = -g\sin\theta. \qquad (5)$$

$$\dot{x} = \chi^{\cdot} + \rho(\cos\theta)\,\theta^{\cdot} + (\sin\theta)\,\rho^{\cdot}$$

$$\ddot{x} = \chi^{\cdot\cdot} + \rho\theta^{\cdot}(-\sin\theta)\theta^{\cdot} + (\cos\theta)(\rho\theta^{\cdot\cdot}+\theta^{\cdot}\rho^{\cdot}) + (\sin\theta)\rho^{\cdot} + \rho^{\cdot}(\cos\theta)\theta^{\cdot}$$

$$\ddot{x} = \chi^{\cdot\cdot} - \rho\theta^{\cdot 2}\sin\theta + (\cos\theta)(\rho\theta^{\cdot\cdot}+\theta^{\cdot}\rho^{\cdot}) + \rho^{\cdot\cdot}\sin\theta \;\; 0 + \rho^{\cdot}\theta^{\cdot}\cos\theta \qquad (\rho^{\cdot\cdot} = 0 \text{ ;ให้ความเร็วในการดึงเชือกคงที่})$$

$$\ddot{x} = \chi^{\cdot\cdot} - (\rho\sin\theta)\theta^{\cdot 2} + 2\rho^{\cdot}\theta^{\cdot}\cos\theta + \rho^{\cdot}\theta^{\cdot}\cos\theta$$

$$\dot{y} = -[\rho(-\sin\theta)\theta^{\cdot} + (\cos\theta)\rho^{\cdot}]$$

$$\ddot{y} = -[\rho\theta^{\cdot}(-\cos\theta)\theta^{\cdot} + (-\sin\theta)(\rho\theta^{\cdot\cdot} + \theta^{\cdot}\rho^{\cdot}) + (\cos\theta)\rho^{\cdot\cdot} + \rho^{\cdot}(-\sin\theta)\theta^{\cdot}]$$

$$\ddot{y} = \rho\theta^{\cdot 2}\cos\theta + \rho\theta^{\cdot\cdot}\sin\theta + 2\rho^{\cdot}\theta^{\cdot}\sin\theta$$

จาก $\ddot{x}\cos\theta + \ddot{y}\cos\theta = -g\sin\theta$

ได้ว่า $\chi^{\cdot\cdot}\cos\theta - (\rho\sin\theta\cos\theta)\theta^{\cdot 2} + 2\rho^{\cdot}\theta^{\cdot}\cos^{2}\theta + \rho\theta^{\cdot\cdot}\cos^{2}\theta + \rho\theta^{\cdot 2}\cos\theta\sin\theta + \rho\theta^{\cdot\cdot}\sin^{2}\theta + 2\rho^{\cdot}\theta^{\cdot}\sin^{2}\theta = -g\sin\theta$

สมมุติให้การแกว่งของมุม $\theta$ มีค่าน้อย ๆ ทำให้เราสามารถประมาณค่า $\sin\theta$ เป็น 0 และ $\cos\theta$ เป็น 1 ทำให้ได้สมการดังนี้

$$\chi^{\cdot\cdot} - \rho\theta\theta^{\cdot 2} + 2\rho^{\cdot}\theta^{\cdot} + \rho\theta^{\cdot\cdot} + \rho\theta^{\cdot 2}\theta + \rho\theta^{\cdot\cdot}\theta^{2} + 2\rho^{\cdot}\theta^{\cdot}\theta^{2} = -g\theta$$

$$\chi^{\cdot\cdot} + 2\rho^{\cdot}\theta^{\cdot}[1+\theta^{2}] + g\theta = -\rho\theta^{\cdot\cdot} - \rho\theta^{\cdot\cdot}\theta^{2}$$

$$[-\chi^{\cdot\cdot}/\rho(1+\theta^{2})] - g\theta/\rho(1+\theta^{2}) - 2\rho^{\cdot}\theta^{\cdot}/\rho = \theta^{\cdot\cdot}$$

เนื่องจาก คิด $\theta$ เป็นค่าน้อย ๆ $(1+\theta^{2})$ จึงประมาณให้เท่ากับ 1

ได้ว่า

$$-(1/\rho)\chi^{\cdot\cdot} - (2\rho^{\cdot}/\rho)\theta^{\cdot} - (g/\rho)\theta = \theta^{\cdot\cdot}$$

# บทที่3

## การควบคุมการแกว่งและตำแหน่งของเครน

ในการควบคุมการเคลื่อนที่ของเครนเราต้องการควบคุมตัวแปรหรือเอาท์พุทของระบบถึงสองค่าคือ ต้องการควบคุมตำแหน่งของเครนและต้องการควบคุมมุมการแกว่งของภาระให้มีค่าน้อยที่สุด เราจึงเลือกวิธีการควบคุมโดยการป้อนกลับสเตทแบบสเตทเรกกูเรเตอร์(State Regulator) ที่หาค่ากฎการควบคุมมาจากการนำสเตทที่ป้อนกลับ มาคูณด้วยค่าเกนป้อนกลับ K (feedback gain, K) ซึ่งทำการออกแบบค่าเกนป้อนกลับ K โดยวิธีการควบคุมออพติมัลแบบเรกกูเรเตอร์เชิงเส้นที่สถานะอยู่ตัว (Steady State Linear Quadratic Regulator) ที่มีดัชนีการทำงานแบบควอดเดรติกเป็น (Quadratric Performance Index) เพื่อให้ได้ผลตอบสนองดีที่สุดภายใต้ค่าดัชนีการทำงานที่ตั้งไว้ แต่เนื่องจากการควบคุมแบบการป้อนกลับสเตทนี้เราจำเป็นต้องเข้าถึงหรือรู้ค่าสเตททุก ๆ ตัวที่เราสนใจที่ประกอบไปด้วย 1. มุมการแกว่งของภาระ($\theta$) 2.ความเร็วเชิงมุมของภาระ($\dot{\theta}$) 3.ตำแหน่งของเครน ($\chi$) 4.ความเร็วของเครน ($\dot{\chi}$ )5.ความเร่งของเครน ($\ddot{\chi}$) ซึ่งในทางปฏิบัติเราทำการวัดมาได้เพียงสองสเตทเท่านั้น คือ มุมการแกว่ง และ ตำแหน่งของเครนดังนั้นเราจึงจำเป็นต้องทำการประมาณค่าสเตทที่เหลือจากค่าสเตทที่เรารู้โดยเราเลือกมา โดยวิธีทำการประมาณค่าสเตทโดยอาศัยความสัมพันธ์ทางฟิสิกส์

### 3.1 การควบคุมออพติมัล (Optimal Control)

การออกแบบระบบควบคุมออพติมัล คือการหากฎควบคุม (control law) เพื่อกำหนดการตัดสินใจสำหรับควบคุมในเวลาปัจจุบัน โดยมีเงื่อนไขบังคับบางประการเพื่อทำให้ระบบเบี่ยงเบนไปจากลักษณะที่ต้องการน้อยที่สุด ตัววัดการทำงานของระบบให้ได้ตามต้องการนี้เรียกว่าเกณฑ์การตัดสินใจหรือ ดัชนีการทำงาน (Performance Index , J) ซึ่งเป็นฟังชั่นที่เราเลือกให้เป็นตัวชี้ว่าการทำงานจริงของระบบใกล้เคียงกับการทำงานที่ต้องการมากน้อยเพียงใด ดังนั้นปัญหาการควบคุมออพติมัลจึงอยู่ที่ การเลือกดัชนีการทำงาน และกฎการควบคุมออพติมัล (Optimal control law , $u_k$ ) เพื่อออพติไมซ์ (optimize) ฟังชั่นดัชนีการทำงาน หรือส่วนใหญ่จะเป็นการหาค่ากฎการควบคุมที่ทำให้ดัชนีการทำงานมีค่าน้อยที่สุด (minimization) นั่นเอง สำหรับการเลือกดัชนีการทำงานนั้นขึ้นกับแต่ละปัญหาเราไม่สามารถกำหนดกฎเกณฑ์ ในการเลือกดัชนีการทำงานให้ตายตัวได้

กำหนดระบบที่สามารถควบคุมได้โดยสมบูรณ์

$$x_{k+1} = Ax_k + Bu_k \quad \text{----------(3.1)}$$

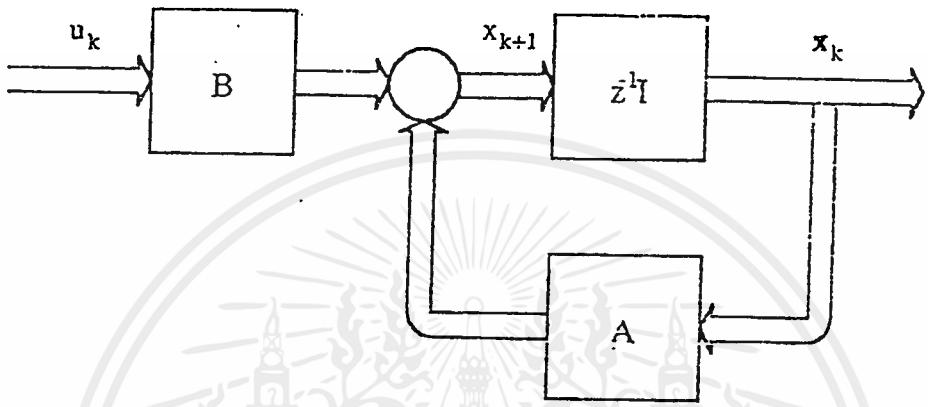
เมื่อ $x_k$ เป็น n สเตทเวคเตอร์

$u_k$ เป็น r เวคเตอร์ควบคุม

A เป็น(nxn) เมตริกซ์

B เป็น(nxr) เมตริกซ์

T หมายถึงทรานสโพส




รูปที่ 3.1แสดงบล็อกไดอะแกรมแบบของระบบ

ในที่นี้เราเลือกใช้ การควบคุมแบบออพติมัลแบบ เรกกูเรเตอร์เชิงเส้นที่สถานะอยู่ตัว (Steady State Linear Quadratric Regulator) ที่มีดัชนีการทำงานควอดเดรติกเป็น

$$J=(1/2)\sum_{k=0}^{\infty}[x_k^T Q x_k+u_k^T R u_k] \qquad \text{----------(3.2)}$$

เมื่อ Q เป็น (nxn) เมตริกซ์เฮอร์มิเชียน (Hermitian Matrix)(หรือ เมตริกซ์สมมาตรเลขจริง(Real Symmetric Matrix)) positive definite หรือ positive semidefinite

R เป็น (rxr) เมตริกซ์เฮอร์มิเชียน (หรือ เมตริกซ์สมมาตรเลขจริง) positive definite

เนื่องจากส่วนประกอบของระบบในทางปฏิบัติจะมีข้อจำกัดในทางกายภาพ จึงต้องมีการกำหนดเงื่อนไขบังคับกับตัวแปรสเตท ตัวแปรควบคุม และสัญญาณควบคุมเพื่อจำกัดขนาดตัวแปรสเตทและขนาดสัญญาณควบคุม เมตริกซ์ Q และ R จะถูกเลือกเพื่อให้น้ำหนักความสำคัญของสเตทและสัญญาณอินพุท แต่ละตัวต่าง ๆกัน เมตริกซ์ Q จะถูกเลือกเพื่อจำกัดค่าผิดพลาดหรือขนาดของสเตทแต่ละตัว ยิ่งเราให้น้ำหนักความสำคัญกับสเตทไหนมาก (เมื่อเปรียบเทียบกับที่ให้กับสเตทอื่น) ค่าผิดพลาดหรือขนาดของสเตทนั้นก็จะถูกจำกัดให้มีค่าน้อยลงเมื่อเทียบ

กับสเตทอื่นและเมตริกซ์ R จะถูกเลือกเพื่อจำกัดขนาดของสัญญาณควบคุมแต่ละตัว เช่นเดียวกันถ้าเราให้น้ำหนักความสำคัญกับอินพุทตัวไหนมาก (เมื่อเทียบกับที่ให้กับอินพุทอื่น) ขนาดของอินพุทนั้นก็จะถูกจำกัดให้มีค่าน้อยลงเมื่อเทียบกับอินพุทตัวอื่นการหาค่ากฎควบคุมออพติมัลนั้น สามารถหาได้หลายวิธีในที่นี้เราจะแสดงการหากฎควบคุมออพติมัล โดยการใช้ฟังชั่นลีอาพูนอฟ (Liapunov Function)

3.1.5 การหากฎการควบคุมออพติมัลสำหรับเรกกูเรเตอร์เชิงเส้นโดยใช้ฟังชั่นลีอาพูนอฟ

จากระบบที่สามารถควบคุมได้โดยสมบูรณ์(สมการ (3.1))

$$x_{k+1}=Ax_k+Bu_k$$

ต้องการหาเมตริกซ์เกน K สำหรับกฎควบคุมออพติมัล

$$u_k = -Kx_k \qquad \text{----------(3.3)}$$

เมื่อ $u_k$ คือกฎควบคุมออพติมัล

ที่ทำให้ดัชนีการทำงานควอดเดรติก(สมการ(3.2))ต่อไปให้น้อยที่สุด

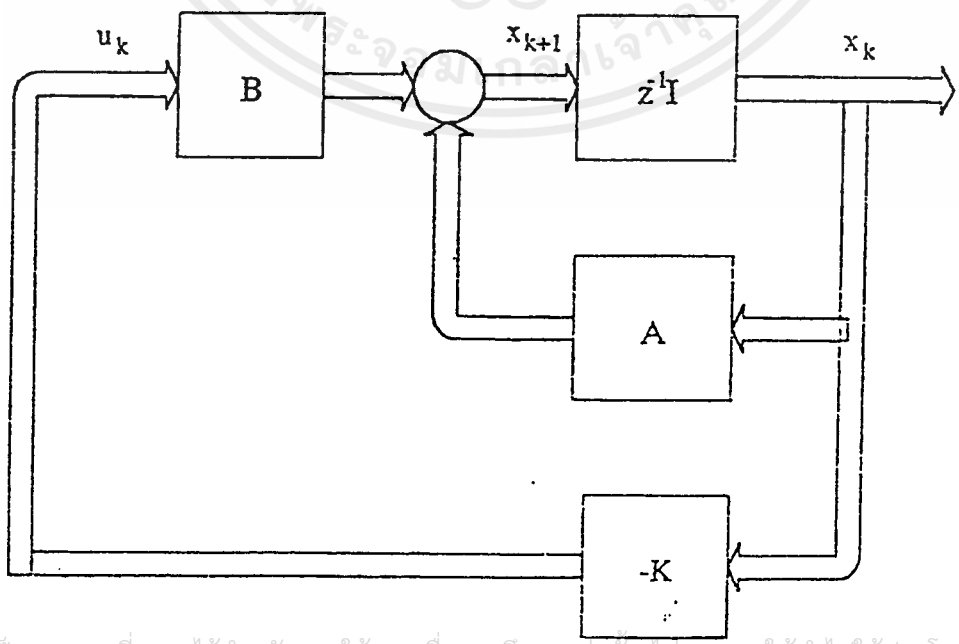
$$J=(1/2)\sum_{k=0}^{\infty} [x_k^T Q x_k+u_k^T R u_k]$$

เนื่องจาก Q เป็นเมตริกซ์เฮอร์มิเชียน positive definite หรือ positive semidefinite และ R เป็นเมตริกซ์เฮอร์มิเชียน positive definite ดังนั้น ดัชนีการทำงาน J จึงเป็น positive definite

เราหาสมการของระบบวงปิดด้วยการป้อนกลับ $u_k = -Kx_k$ ได้

$$x_{k+1} = (A-BK)x_k \qquad \text{----------(3.4)}$$

$$J=(1/2)\sum_{k=0}^{\infty} [x_k^T (Q+K^T R K)x_k] \qquad \text{----------(3.5)}$$




รูปที่ 3.2 รูปแสดงบล็อกไดอะแกรมของระบบเมื่อป้อนกลับด้วย $u_k=-Kx_k$

ในที่นี้ให้ A-BK เป็นเมตริกซ์เสถียร ดังนั้นย่อมหาฟังก์ชั้นลีอาพูนอฟ ที่เป็น positive definite ได้และมีอนุพันธ์อันดับหนึ่งเป็น negative definite ให้ฟังก์ชั่นลีอาพูนอฟคือ

$$V(x_k)=x_k^T P z_k$$

เมื่อ P เป็นเมตริกซ์เฮอร์มิเชียน (หรือสมมาตรเลขจริง ) positive definite อนุพันธ์อันดับหนึ่งของ $V(x_k)$ คือ

$$\Delta V(x_k) = V(x_{k+1})-V(x_k)$$

ได้ $$\Delta V(x_k) = x_{k+1}^T P_{k+1} - x_k^T P x_k$$

จากสมการ (3.1)

$$\Delta V(x_k) = x_k^T[(A-BK)^T P(A-BK) - P]x_k$$

ซึ่งเป็น negative difinite และ $x_k^T(Q+K^T RK)x_k$ เป็น positive definite เราจึงให้

$$x_k^T(Q+K^T RK)x_k = -x_k^T[(A-BK)^T P(A-BK)-P]x_k$$

สมมติว่าสมการดังกล่าวเป็นจริงสำหรับทุกค่า $x_k$ จะได้

$$Q+K^T RK =-(A-BK)^T P(A-BK)+P$$

$$Q+A^T PA-P+K^T(R+B^T PB)K-(K^T B^T PA+A^T PBK) =0$$

$$Q+A^T PA-P+[(R+B^T PB)^{1/2}K-(R+B^T PB)^{-1/2} B^T PA]^T[(R+B^T PB)^{1/2} K-(R+B^T PB)^{-1/2}B^T P A]-A^T PB(R+B^T PB)-1B^T PA =0 \quad \text{----------(3.6)}$$

การทำให้ดัชนีการทำงานควอดเดรติก น้อยที่สุดโดยคิดจาก เกนการป้อนกลับ K ก็คือการเลือกค่าเกน K ที่ทำให้ด้านซ้ายมือของสมการ(3.6) มีค่าน้อยที่สุด และ เนื่องจากเทอม $[(R+B^T PB)^{1/2}K-(R+B^T PB)^{-1/2} B^T PA]^T[(R+B^T PB)^{1/2} K-(R+B^T PB)^{-1/2}B^T P A]$ ไม่เป็นลบเสมอ ดังนั้นจึงจะมีค่าน้อยที่สุด เมื่อเป็นศูนย์เท่านั้น หรือเมื่อ

$$(R+B^T PB)^{1/2} K=(R+B^T PB)^{-1/2} B^T PA$$

$$K=(R+B^T PB)^{-1} B^T PA \quad \text{----------(3.7)}$$

แทนค่า เกน K ที่ได้จากสมการ (3.7) ลงในสมการ (3.6) จะได้ค่า P ที่ทำให้ดัชนีการทำงานควอดเดรติกมีค่าน้อยที่สุด

$$P=Q+A^T PA-A^T PB(R+B^T PB)^{-1} B^T PA \quad \text{----------(3.8)}$$

สมการ(3.8) นี้ เรียกว่า สมการริคคาติ (Riccati Equation) สถานะอยู่ตัว

จากสมการ (37) และสมการ (3.8) เช่นเดียวกัน จะเห็นได้ว่าค่าของเมตริกซ์ P และของเมตริกซ์ K จะขึ้นกับค่าของเมตริกซ์ Q และเมตริกซ์ R ด้วย ดังนั้นถ้าสามารถกำหนดเมตริกซ์ Q และเมตริกซ์ R ได้เหมาะสมกับระบบแล้ว เราก็จะได้ค่า เกนการป้อนกลับ K ที่ทำให้ระบบมีผลตอบสนองดีที่สุด

จากสมการ(3.7) และสมการ (3.8) ที่ได้แสดงให้เห็นว่า ถ้าระบบ สามารถควบคุมได้โดยสมบูรณ์ และเราสามารถวัดค่าจากทุกสเตทได้ เมื่อเราหาโมเดลของระบบได้ และกำหนดเมตริกซ์ Q และ เมตริกซ์ R เพื่อให้น้ำหนักความสำคัญกับสเตทและอินพุทแต่ละตัวแล้ว เราจะสามารถแก้สมการ (3.8) หาค่าเมตริกซ์ P ได้ และนำค่าเมตริกซ์ P ไปหาค่า เกน K ป้อนกลับสำหรับระบบเพื่อนำไปหากฎควบคุม ณ เวลาต่าง ๆ ได้

3.2 การประมาณค่าสเตทโดยอาศัยความสัมพันธ์ทางฟิสิกส์

การประมาณค่าสเตททำได้ดังนี้

1.การประมาณค่าความเร็วเชิงมุมของภาระ ($\theta^{\cdot}$) กับค่ามุมการแกว่งของภาระ($\theta$)

$\theta^{\cdot}$ = อัตราการเปลี่ยนแปลงของมุมการแกว่งของภาระ($\theta$)ในหนึ่งหน่วยเวลา

ดังนั้นความเร็วเชิงมุมของภาระในช่วงคาบเวลาการชักตัวอย่างที่ k ($\theta^{\cdot}_k$)จะมีค่าเป็น

$$\theta^{\cdot}_k = (\theta_k - \theta_{k-1})/(\text{sampling time}) \qquad \text{----------(3.9)}$$

เมื่อ $\theta_k$ คือค่ามุมการแกว่งที่วัดได้ในการชักตัวอย่างที่ k

$\theta_{k-1}$ คือค่ามุมการแกว่งที่วัดได้ในการชักตัวอย่างที่ k-1

sampling time คือค่าคาบเวลาที่ใช้ในการชักตัวอย่าง

2.การประมาณค่าความเร็วเครน

จากความสัมพันธ์ของความเร็วของเครน ($\chi^{\cdot}$) กับตำแหน่งของเครน ($\chi$)

$\chi^{\cdot}$ = อัตราการเปลี่ยนแปลงของตำแหน่งของเครน ($\chi$ )ในหนึ่งหน่วยเวลา

ดังนั้นความเร็วของเครนในช่วงคาบเวลาการชักตัวอย่างที่ k ($\chi^{\cdot}_k$)จะมีค่าเป็น

$$\chi^{\cdot}_{k-1} = (\chi_k - \chi_{k-1})/(\text{sampling time}) \qquad \text{----------(3.10)}$$

เมื่อ $\chi_k$ = คือค่าตำแหน่งของเครนที่วัดได้ในการชักตัวอย่างที่ k

$\chi_{k-1}$ = คือค่าตำแหน่งของเครนที่วัดได้ในการชักตัวอย่างที่ k-1

3.การประมาณค่าความเร่งของเครน

จากความสัมพันธ์ของความเร่งของเครน ($\chi^{\cdot\cdot}$)กับความเร็วของเครน($\chi^{\cdot}$)

$\chi^{\cdot\cdot}$ = อัตราการเปลี่ยนแปลงของความเร็วของเครน ($\chi^{\cdot}$)ในหนึ่งหน่วยเวลา

ดังนั้นความเร่งของเครนในช่วงคาบเวลาการชักตัวอย่างที่ k ($\chi^{\cdot\cdot}_k$)จะมีค่าเป็น

$$\chi^{\cdot\cdot}_k = (\chi^{\cdot}_k - \chi^{\cdot}_{k-1})/(\text{sampling time}) \qquad \text{----------(3.11)}$$

เมื่อ $\chi^{\cdot}_k$ คือค่าความเร็วของเครนที่วัดได้ในการชักตัวอย่างที่ k

$\chi^{\cdot}_{k-1}$ คือค่าความเร็วของเครนที่วัดได้ในการชักตัวอย่างที่ k-1

# บทที่4

## การประมาณค่าโมเดลของระบบจากผลตอบสนองที่ได้การทดลอง

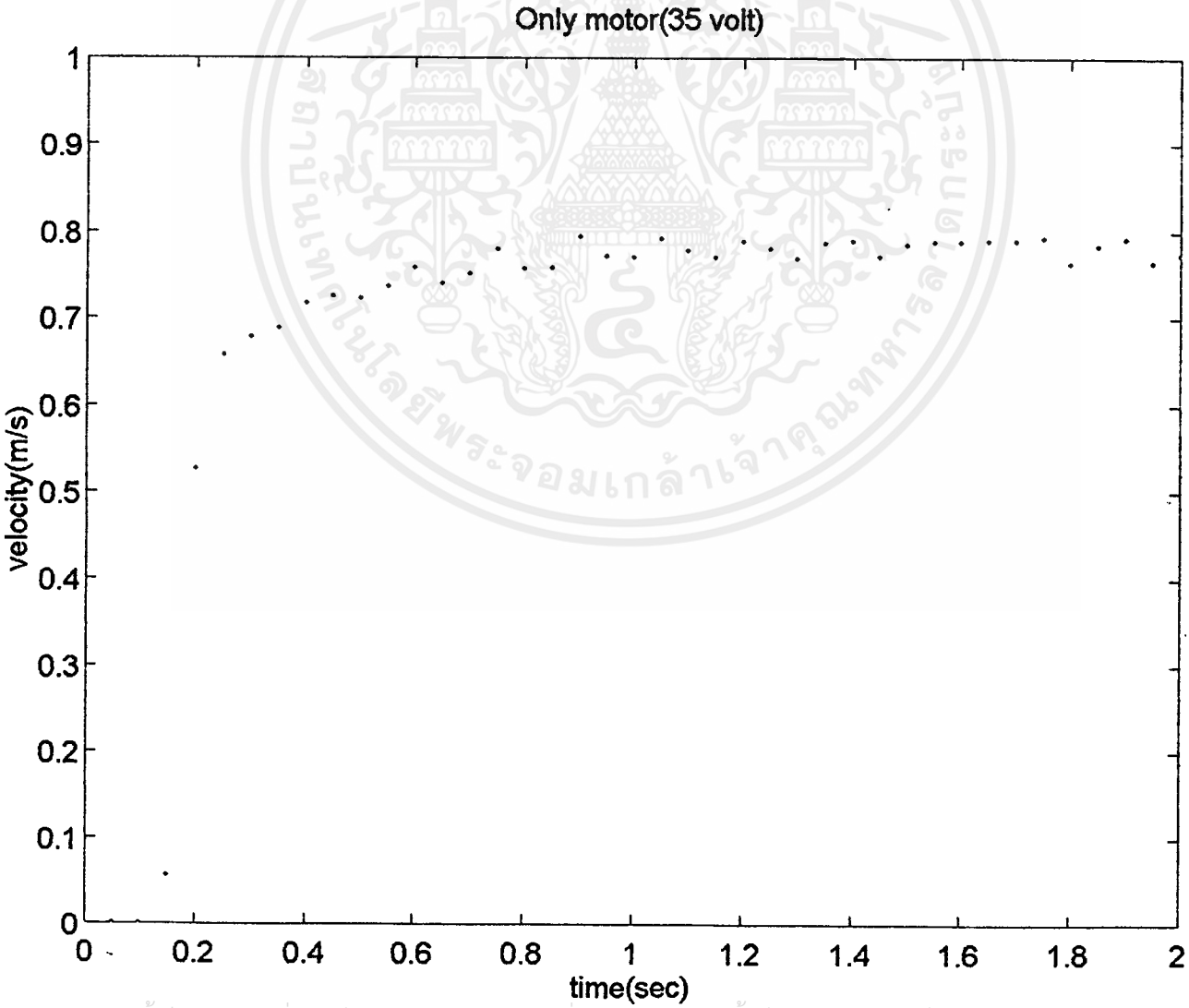
### 4.1 การหาโมเดลของมอเตอร์

จากการพิจารณา ผลตอบสนองของมอเตอร์ต่อ Step Input 35 V ของมอเตอร์ตามรูปที่ 4.1 นั้นสามารถพิจารณาได้ว่าระบบเป็นระบบอันดับสองดังสมการข้างล่าง

$$\frac{C(s)}{R(s)} = \frac{K\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \qquad \text{.....(4.1)}$$

และมีผลตอบสนองต่อเวลาเป็น

$$c(t) = K\left(1 - \frac{e^{-\zeta\omega_n t}}{\sqrt{1-\zeta^2}} \sin\left(\omega_d t + \tan^{-1}\frac{\sqrt{1-\zeta^2}}{\zeta}\right)\right) \qquad \text{.....(4.2)}$$




รูปที่ 4.1 ผลตอบสนองความเร็วต่อ Step Voltage

จากผลตอบสนองที่ได้จากการทดลอง ให้ Step input 35 V เราสามารถประมาณค่าพารามิเตอร์ต่าง ๆของระบบได้โดยจะเห็นได้ว่าที่35 V สามารถ ให้ความเร็วสูงสุด 0.8 เมตร ต่อ วินาที หลังจากนั้นเราก็จะสามารถหา พารามิเตอร์ของ มอเตอร์รวมกับLM629ได้ดังต่อไปนี้

4.2 การหาโมเดลของ LM629 รวมกับมอเตอร์

เราทดลองป้อนคำสั่ง Step Speed 0.8 m/s จาก Computer ให้ LM629 แล้ววัดผลตอบสนองทางความเร็วได้ดังรูปที่ 4.1 จากผลตอบสนองทางความเร็วที่ได้ สามารถประมาณได้เป็นระบบอันดับสอง เราสามารถประมาณค่าพารามิเตอร์ต่าง ๆ ของระบบได้โดยปรับค่า $\zeta$ และ $\omega_n$ เพื่อให้ได้ผลตอบสนองที่ใกล้เคียงกับผลตอบสนองที่ได้จากระบบจริงมากที่สุด โดยพิจารณาจากกราฟผลตอบสนองของระบบจริงกับผลตอบสนองที่ได้จากการ Simulate บน MATLAB ซึ่งได้ผลดังนี้

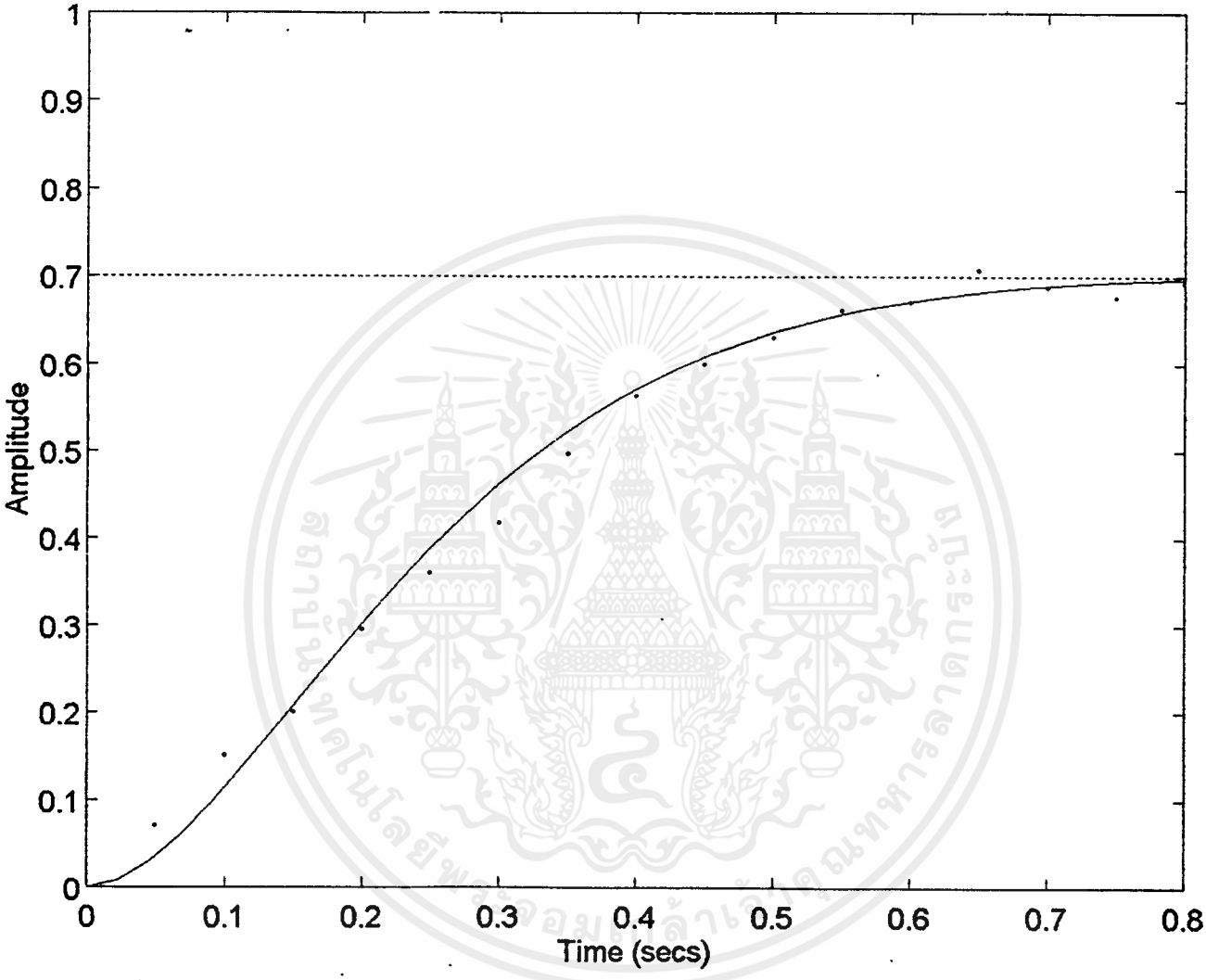
$$\frac{C(s)}{R(s)} = \frac{K\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \qquad \text{----------(4.3)}$$

K = 0.8750 ; $\omega_n$ = 7 rad/sec ; $\zeta$ = 0.9

C(s) = ความเร็วของตัวรถเครน

R(s) = คำสั่งความเร็วที่ส่งมาจาก Computer

ดังรูปในหน้าถัดไป




รูปที่4.2 ผลตอบสนองความเร็วต่อ step voltage เปรียบเทียบกับ model ที่ประมาณได้

# บทที่ 5

## การ Simulation และการทดลอง

ในบทนี้จะแสดงถึงผลที่ได้จากการจำลองการทำงานของระบบ (Simulation) โดยคอมพิวเตอร์ และผลการทดลองที่ได้จากการทดลองจริงซึ่งประมาณค่าสเตทจากความสัมพันธ์ทางฟิสิกส์

5.1การ Simulation

จาก โมเดลทางคณิตศาสตร์ในแบบจำลองเครน

$$-(1/\rho)\chi^{\cdot\cdot} - (2\dot{\rho}/\rho)\theta^{\cdot} - (g/\rho)\theta = \theta^{\cdot\cdot} \qquad \text{---------(5.1)}$$

และจาก สมการ ที่ได้จากการทดสอบมอเตอร์

$$\frac{C(s)}{R(s)} = \frac{K\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2} \qquad \text{---------(5.2)}$$

เมื่อ $\theta(t)$ = มุมการแกว่งของภาระ

$\chi(t)$ = ตำแหน่งของรถเครน

C(s) = ความเร็วของตัวรถเครน

R(s) = คำสั่งความเร็วที่ส่งมาจากคอมพิวเตอร์

g = ความเร่งเนื่องจากแรงโน้มถ่วงของโลก

$\rho(t)$ = ความยาวเชือกที่ใช้แขวนภาระ

จากสมการที่ได้จาก (1) และ (2) มาเขียนโมเดลของระบบ $x = Ax + Bu$ ได้ดังนี้

โดย $x = [\, x_1\ x_2\ x_3\ x_4\ x_5 \,]^T = [\theta\ \theta^{\cdot}\ \chi\ \chi^{\cdot}\ \chi^{\cdot\cdot}]^T$

$$\begin{bmatrix} \theta^{\cdot} \\ \theta^{\cdot\cdot} \\ \chi^{\cdot} \\ \chi^{\cdot\cdot} \\ \chi^{\cdot\cdot\cdot} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & \theta \\ -g/\rho & 2\dot{\rho}/\rho & 0 & 0 & -1/\rho & \theta^{\cdot} \\ 0 & 0 & 0 & 1 & 0 & \chi \\ 0 & 0 & 0 & 0 & 1 & \chi^{\cdot} \\ 0 & 0 & 0 & -\omega_n^2 & -2\zeta\omega_n & \chi^{\cdot\cdot} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ K\omega_n^2 \end{bmatrix} u$$

$$y = [\, 0\ \ 0\ \ 1\ \ 0\ \ 0 \,] \begin{bmatrix} \theta^{\cdot\cdot} \\ \theta^{\cdot} \\ \chi \\ \chi^{\cdot} \\ \chi^{\cdot\cdot} \end{bmatrix}$$

นำพารามิเตอร์ต่าง ๆ ของระบบเครน จากการประมาณ Transfer Function ของ LM629 กับ มอเตอร์ ได้ว่า K=0.8750 $\omega_n = 7$ $\zeta = 0.9$

กำหนดให้เชือกที่ใช้แขวนโหลดมีความยาวที่แปรค่าได้มีค่าเป็น $\rho$ จะได้ $\dot{\rho}$ เป็นความเร็วของการเคลื่อนที่ของเชือก นำค่า พารามิเตอร์ต่าง ๆ แทนในโมเดลของระบบได้ดังนี้

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{\chi} \\ \ddot{\chi} \\ \dddot{\chi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ -9.81/\rho & 2\dot{\rho}/\rho & 0 & 0 & -1/\rho \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -49 & -12.6 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ \chi \\ \dot{\chi} \\ \ddot{\chi} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 42.875 \end{bmatrix} u$$

จากนั้นทำการแปลง State Space Model เวลาต่อเนื่องให้เป็นเวลาดิสครีต โดยให้ sampling time = 0.1 sec โดยใช้ฟังค์ชั่น c2d ในโปรแกรม MATLAB ก็จะสามารถหาค่า G และ H ของระบบดีสครีตได้

```
>>[G,H]=c2d(A,B,0.1)
```

หลังจากที่ได้ Matrix G และ H แล้วนั้น เนื่องจากเราให้ ดัชนีการทำงานของ LQR เป็น

$$J=(1/2)\sum_{k=0}^{\infty} [x_k^T Q x_k + u_k^T R u_k]$$

คำนวณค่า LQR Steady State Gain Matrix , K โดยกำหนดให้

```
>>Q = diag ([1000 1 1000 1 1])   ;R = [200];
```

Q เป็น Weight Matrix ของ X ; R เป็น Weight Matrix ของ u

นำค่า G,H,Q,R ที่ได้ไปคำนวณค่า K ใน MATLAB ได้ โดยใช้ ฟังค์ชั่น dlqr ซึ่งเป็นฟังค์ชั่นที่ ทำหน้าที่หาค่าเกนป้อนกลับ ให้กับ state ตามดัชนีของ LQR

```
>>[ K,S,E ] = dlqr(G,H,Q,R)
```

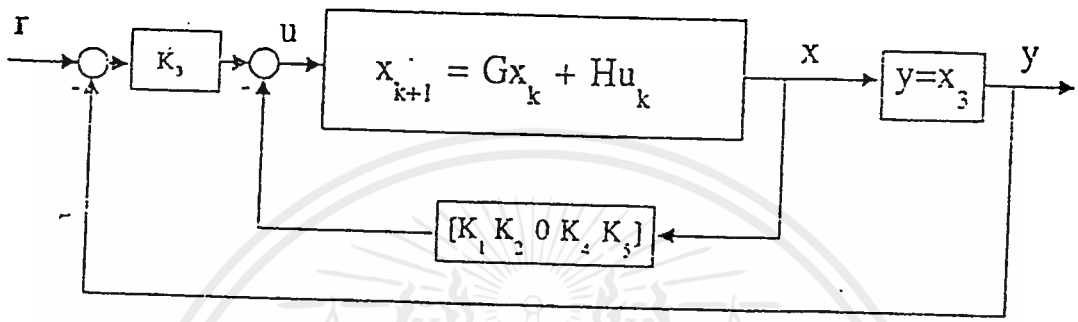
K เป็น Optimal Feedback Gain Matrix ซึ่งจะ minimize ค่า J

S เป็น ผลเฉลยที่ สถานะคงตัวของสมการ Riccati

และ E เป็น Eigenvalue ของระบบ ;E=EIG(G-H*K)

ซึ่งถ้า Eigenvalue ของระบบ อยู่ในวงกลม 1 หน่วย จะบอกให้เราทราบว่าระบบนั้นเสถียร

โครงสร้างการทำงานของระบบควบคุมการเคลื่อนที่ และการแกว่งของโหลดที่แขวนบนรถเครนมีลักษณะเป็นการป้อนกลับสเตท โดยนำทุก ๆ สเตทที่วัดได้ในแต่ละคาบการชักตัวอย่างมาคูณค่าเกน K แล้วนำผลคูณที่ได้ป้อนกลับให้เป็นอินพุท u เพื่อควบคุมให้ทุก ๆ สเตทลู่เข้าสู่ค่าศูนย์ ยกเว้น $x_3$ ซึ่งเป็นตำแหน่งของรถเครนจะลู่เข้าสู่ตำแหน่งอ้างอิง r นั่นก็คือ อินพุทจะลู่เข้าสู่ศูนย์เมื่อ r-y = 0 และ สเตททุก ๆ สเตทเป็นศูนย์โดยสามารถเขียน บล็อกไดอะแกรม การทำงานได้ดังนี้




รูปที่ 5.1 บล็อกไดอะแกรมการทำงานของระบบควบคุม

เราสามารถนำความสัมพันธ์ดัง บล็อกไดอะแกรม ข้างบนมาเขียนเป็นสเตทป้อนกลับรวมได้ดังนี้

$$x_{k+1} = Gx_k + Hu_k$$
$$= Gx_k + H(-Kx_k + k_3 r)$$
$$= (G-HK)x_k + Hk_3 r$$
$$y = Cx_k$$

เนื่องจาก การเคลื่อนที่ของภาระมีการดึงขึ้น ลง เพื่อ หลบหลีกสิ่งกีดขวาง ค่าความยาวเชือกจึงมีค่าไม่คงที่ เปลี่ยนแปลงไปตามเวลา ส่งผลให้ค่าเกนที่คำนวนโดย LQR มีค่าเกนเปลี่ยนไปตามแต่ละความยาวเชือก

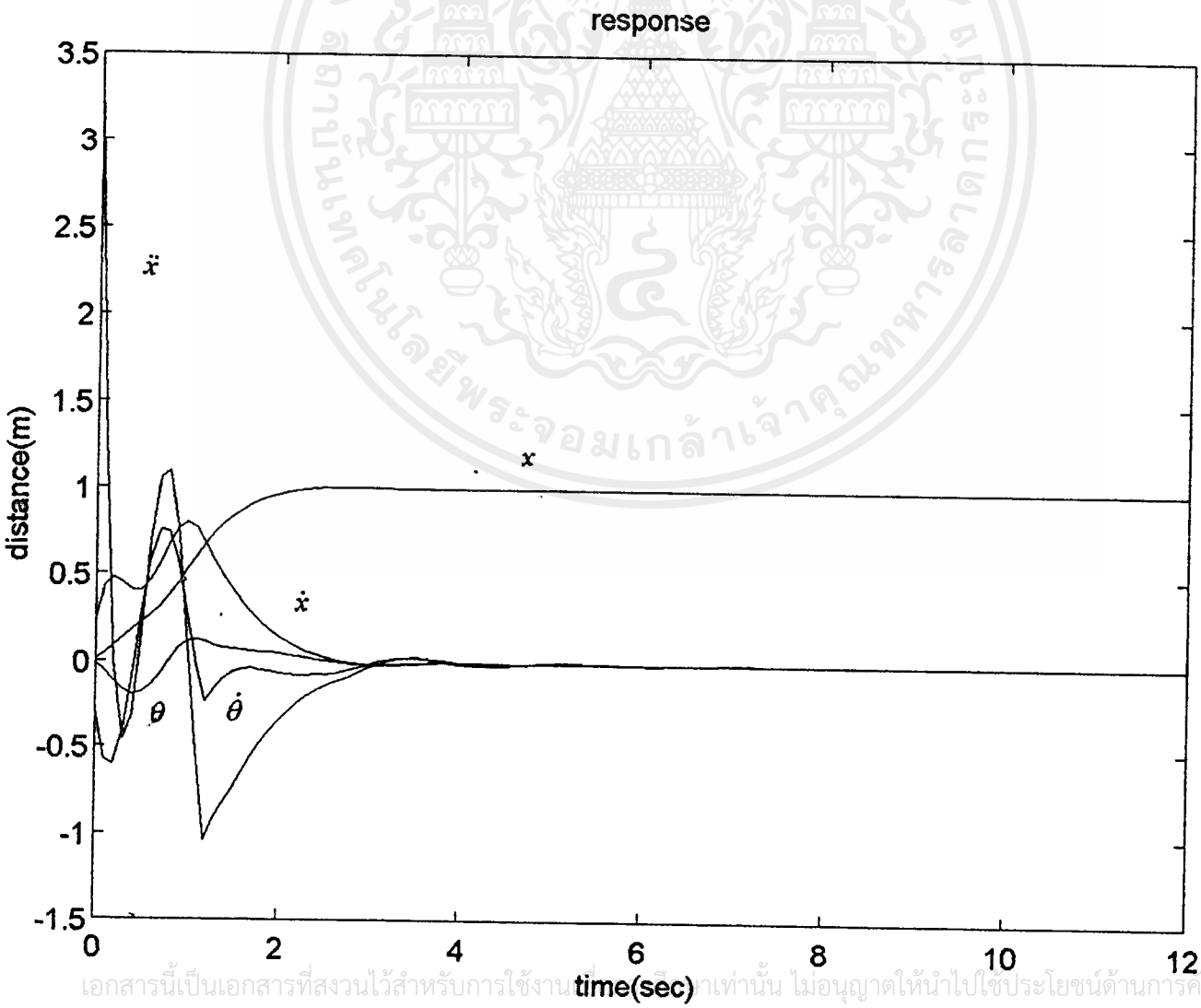
เราจึงนำผลและแนวความคิดที่ได้ ไปเขียนโปรแกรมเพื่อ Simulate เพื่อหาค่าตอบสนองต่อ Step Input โดยพิจารณาว่าเราสามารถวัดค่ามาได้ทุกสเตท และ การคำนวน มีความเร็วพอที่จะหาค่า K เพื่อนำไปป้อนกลับได้ทันเวลาก่อนจะถึงสเตทต่อไป ดังเป็นโปรแกรมดังนี้

```
t=0:0.1:11.9;
L=t;
for n=1:13,
L(n)=-0.5*t(n)+0.8;
end
for n=13:24,
L(n)=0.5*t(n)-0.4;
end
for  n=25:size(t,2)
L(n)=0.8;
end
plot(t,L);
axis([0 3 0 1]);

 difL=0:0.1:11.9;
    difL(1)=0;
for n = 2:size(L,2),
    difL(n)=(L(n)-L(n-1))/0.1;
end
plot(t,difL)
axis([0 6 -1 1])

datafile = fopen('calfrsl5.dat','wt+');
x1=[0 0 0 0 0 ]';r=1;
for n = 1:size(L,2),

   A=[0 1 0 0 0;-9.81/L(n) -2*difL(n)/L(n) 0 0 -1/L(n);0 0 0 1 0;0 0 0 0 1;0 0 0 -49 -12.6];

   B=[0 0 0 0 49*7/8]';
   C=[0 0 1 0 0];

   [G,H] = c2d(A,B,0.1);
   Q=diag([1000 1 1000 1 1]); R=(200);
   [K,S,E]=dlqr(G,H,Q,R);

  x2=((G-H*K)*x1)+((H*K(3))*r);
  fprintf(datafile,'%2.8f  %2.8f  %2.8f  %2.8f  %2.8f \n',x2);
```

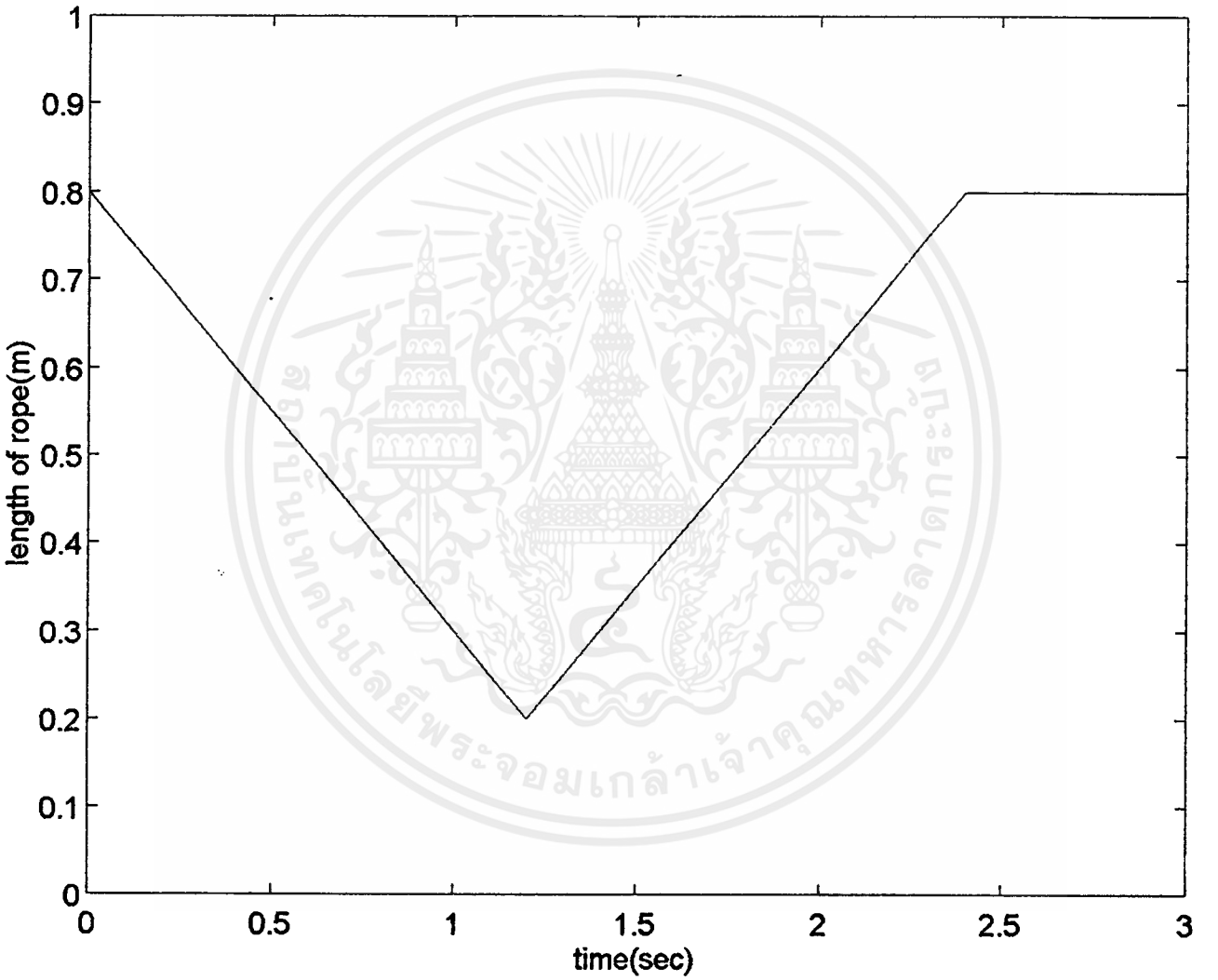
```
x1=x2;
end
fclose(datafile);
load c:\matlab\toolbox\control\calfrsl5.dat;
plot(t,calfrsl5)
```

ซึ่งโปรแกรมอันนี้ เริ่มจากการใส่ค่าของการเคลื่อนที่ของความยาวเชือกซึ่งต้องเขียนเป็นโปรแกรมเข้าไป หลังจากนั้นจึงคำนวนค่า ผลตอบสนองต่อ ฟังก์ชั่นสัญญาณขั้นบันไดหนึ่งหน่วย ที่ละจุดแล้วนำไปเขียนลงไฟล์ โดยแต่ละจุดมีการคำนวณค่า เกนของแต่ละจุดเพื่อให้ ฟังค์ชั่นการคำนวนนั้นสมบูรณ์

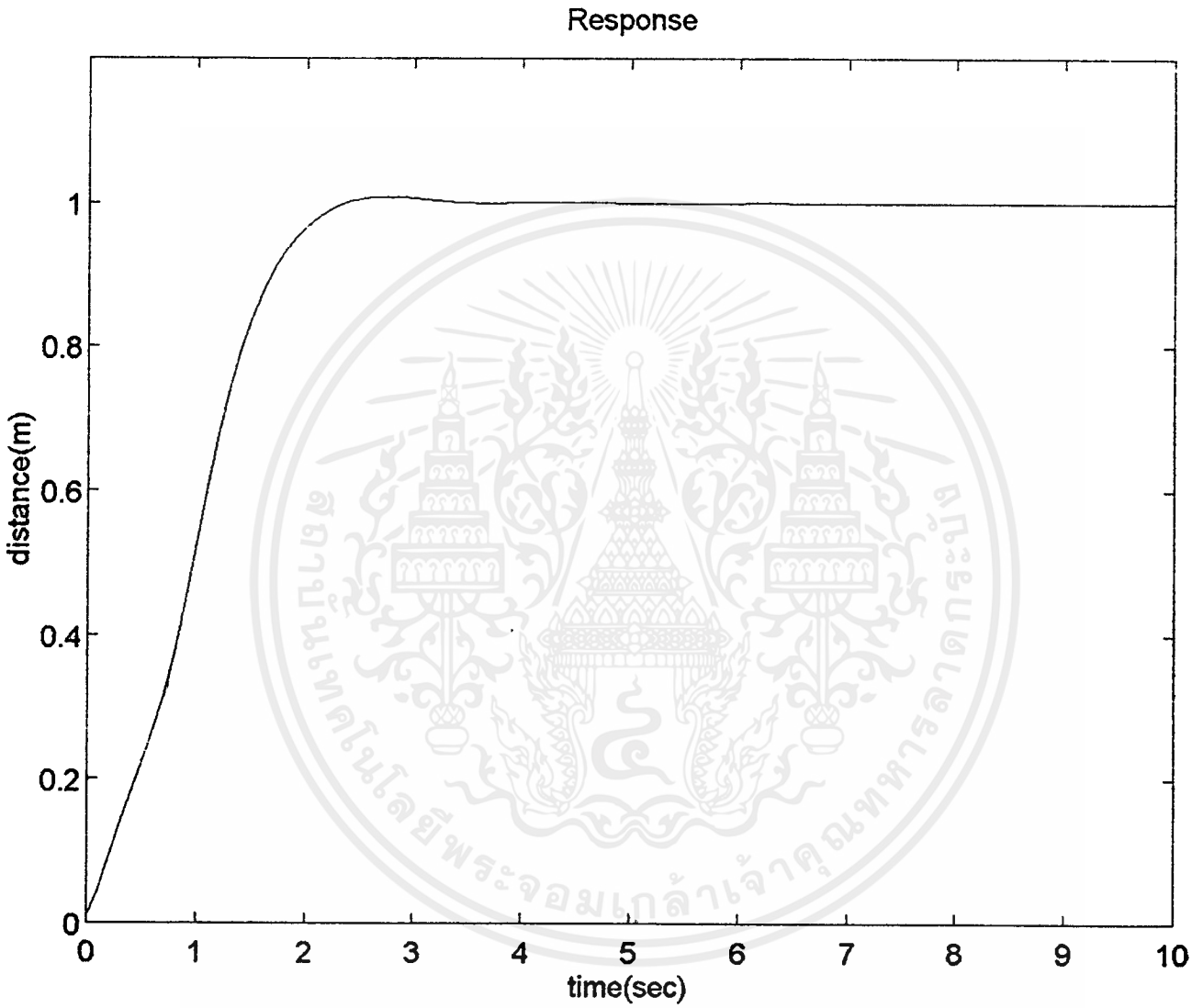
ผลจากการ Simulate ข้างต้นจะได้ผลตอบสนองของระบบดังรูปที่ 5.2ก การเคลื่อนที่ของความยาวเชือกรูปที่ 5.2ข ผลตอบสนองตำแหน่งรถเครนดังรูป ที่ 5.2ค ผลตอบสนองของมุมการแกว่งดังรูปที่ 5.2ง โดยจะได้ว่าทุก ๆ สเตท จะลู่เข้าสู่ศูนย์ยกเว้นเอาท์พุทของระบบที่จะลู่เข้าสู่ค่าตำแหน่งอ้างอิง



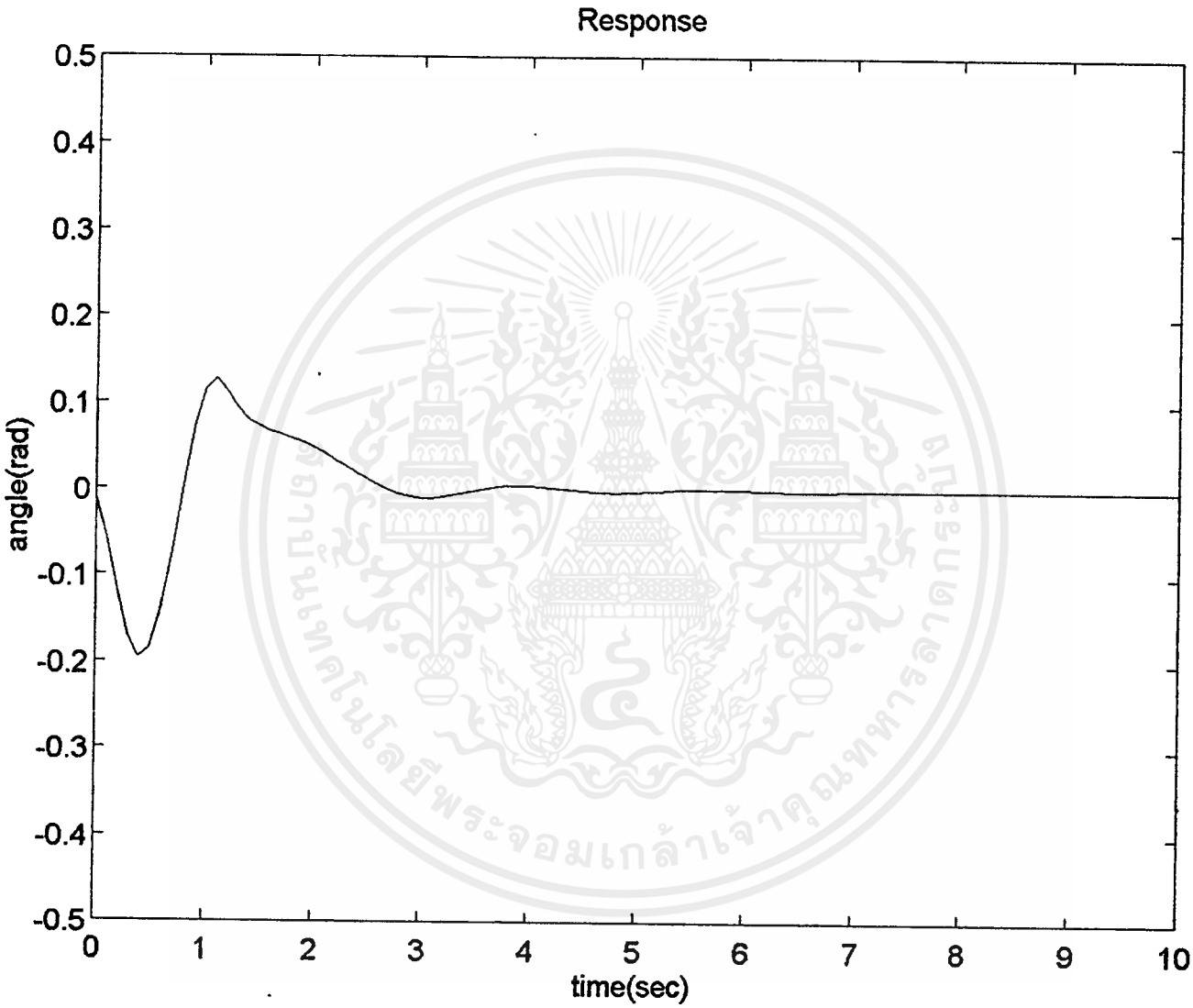

รูปที่5.2ก ผลตอบสนองต่อสเตทต่างๆของระบบเครน




**รูป 5.2ข** การเคลื่อนที่ของความยาวเชือก




**รูป 5.2ค ผลตอบสนองตำแหน่งรถเครน**




รูป 5.2ง ผลตอบสนองของมุมการแกว่ง

5.2 ผลการทดลอง

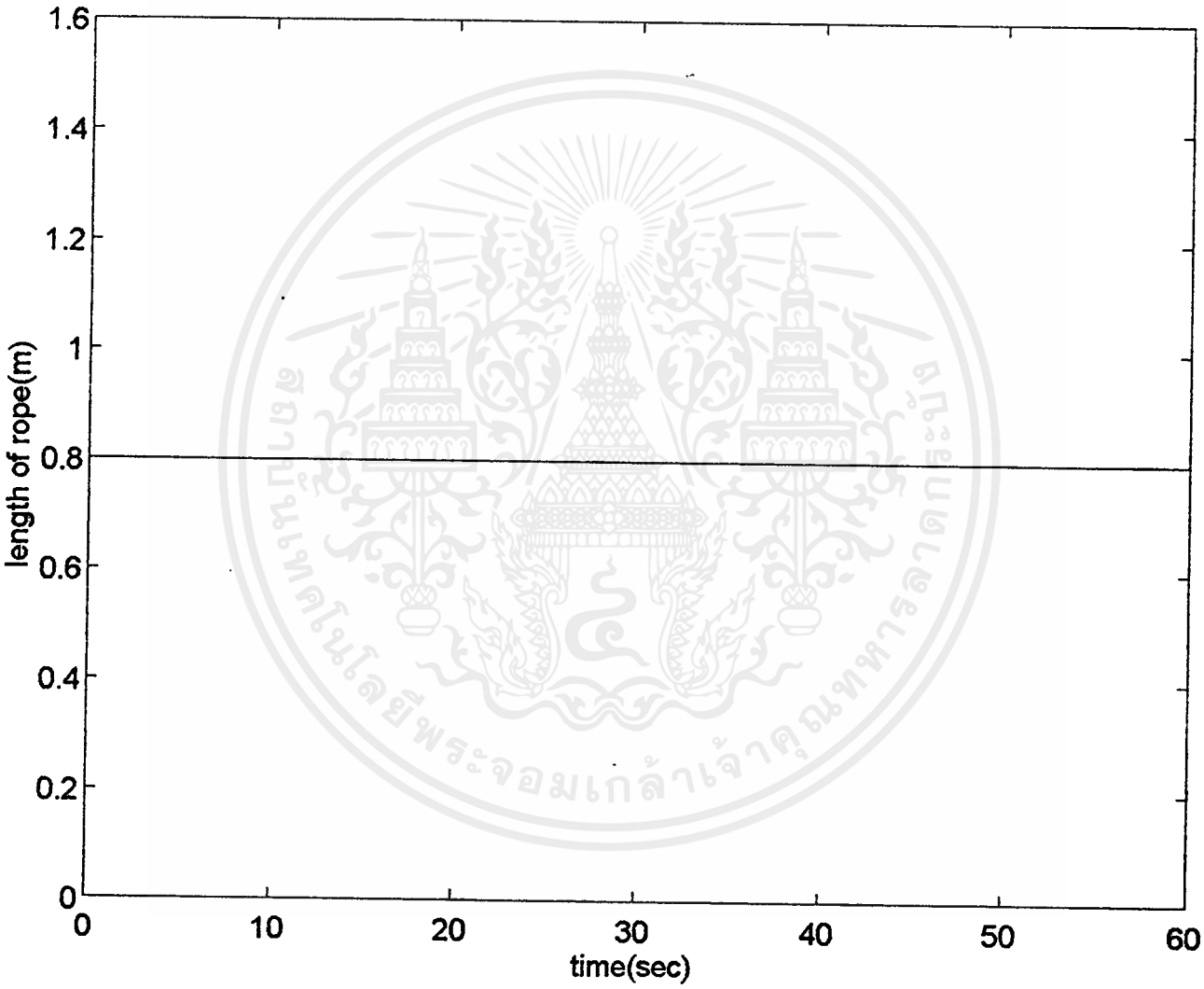
การทดลองเมื่อใช้การประมาณค่าสเตทจากความสัมพันธ์ทางฟิสิกส์ เมื่อทำการทดลองการควบคุมโดยใช้โปรแกรมควบคุมที่ใช้การประมาณค่าสเตทจากความสัมพันธ์ทางฟิสิกส์ และใช้ค่าเกนป้อนกลับ K ที่ออกแบบได้จาก MATLAB จากหัวข้อที่ 5.1 มาคูณกับค่าสเตทต่างๆ ทั้งที่ได้จากการวัดจริงและได้จากการคำนวณ ตามวิธี LQR เพื่อหาค่าสัญญาณควบคุม(ดู source code ของโปรแกรมควบคุมชื่อ work4_2.c ได้ที่ภาคผนวก)

ผลการทดลองแบ่งเป็น 2 ส่วน

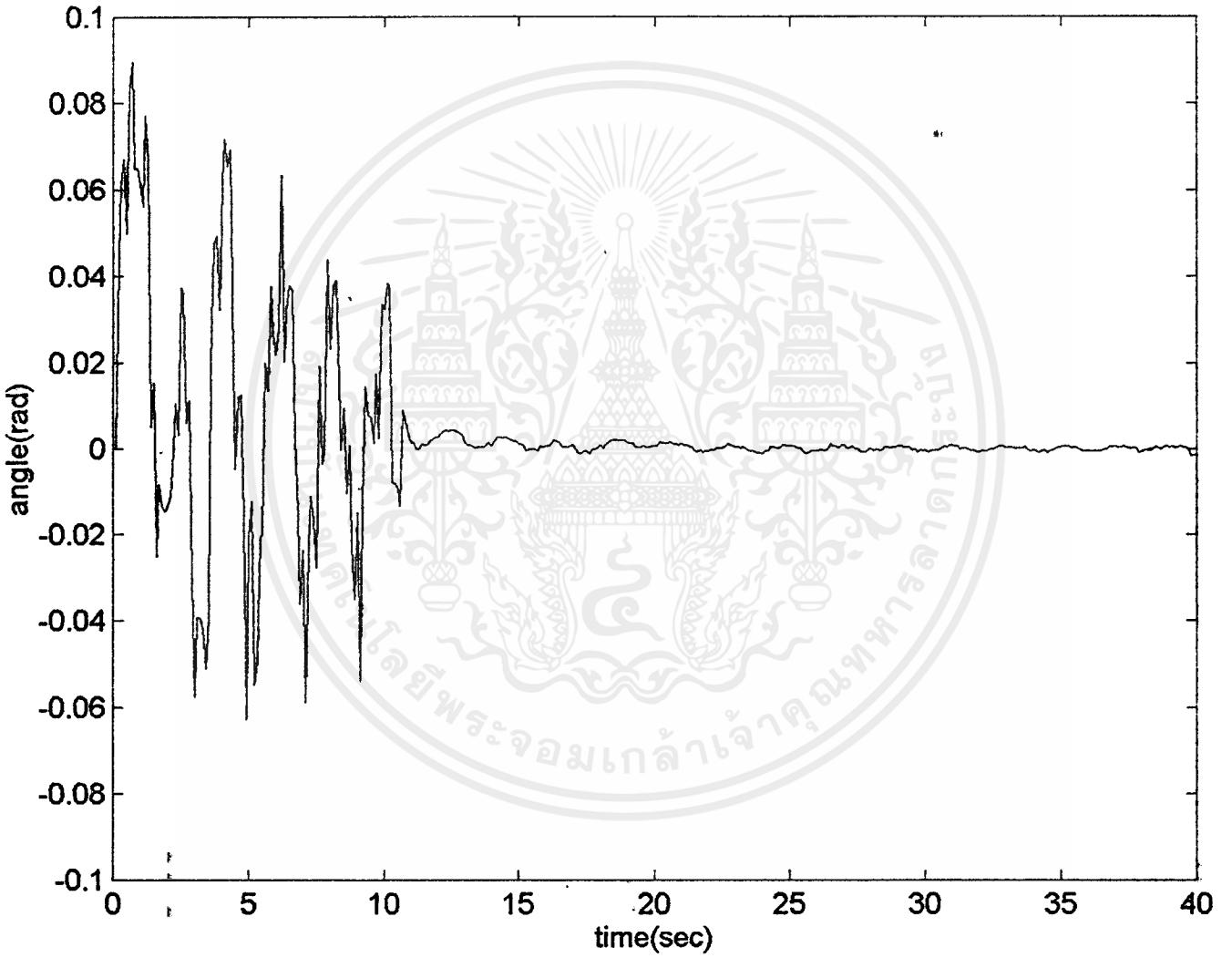
5.2.1 เมื่อความยาวเชือกคงที่ ดังรูปที่ 5.3ก-5.3ฉ

5.2.2 เมื่อความยาวเชือกเปลี่ยนแปลง ดังรูปที่ 5.4ก-5.4ฉ

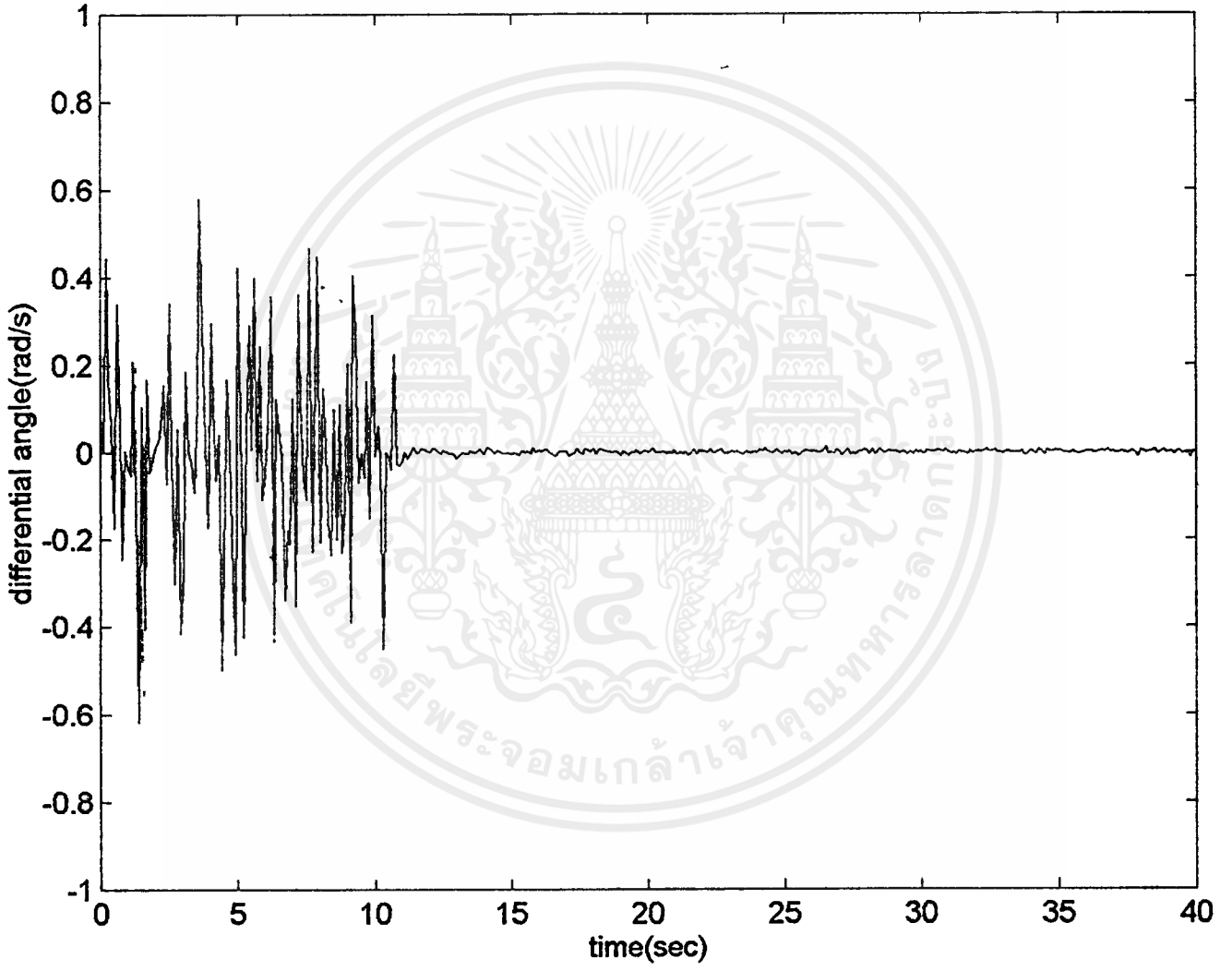
5.2.1 ผลการทดลองการเคลื่อนที่ของเครน เมื่อให้ความยาวเชือกของภาระมีค่าคงที่



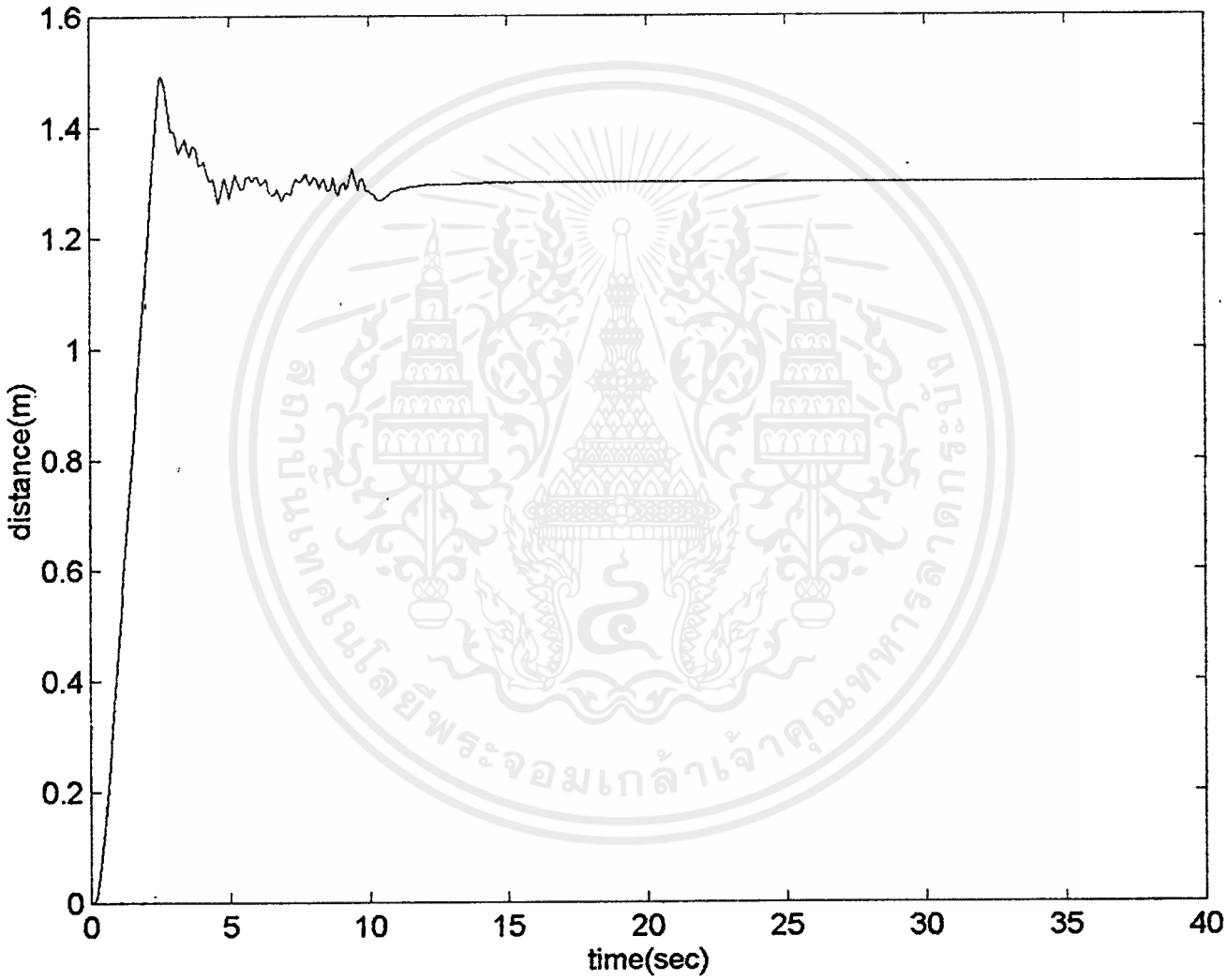

รูปที่ 5.3ก รูปแสดงความยาวเชือกที่เปลี่ยนไปต่อเวลาที่ใช้ในการเคลื่อนที่



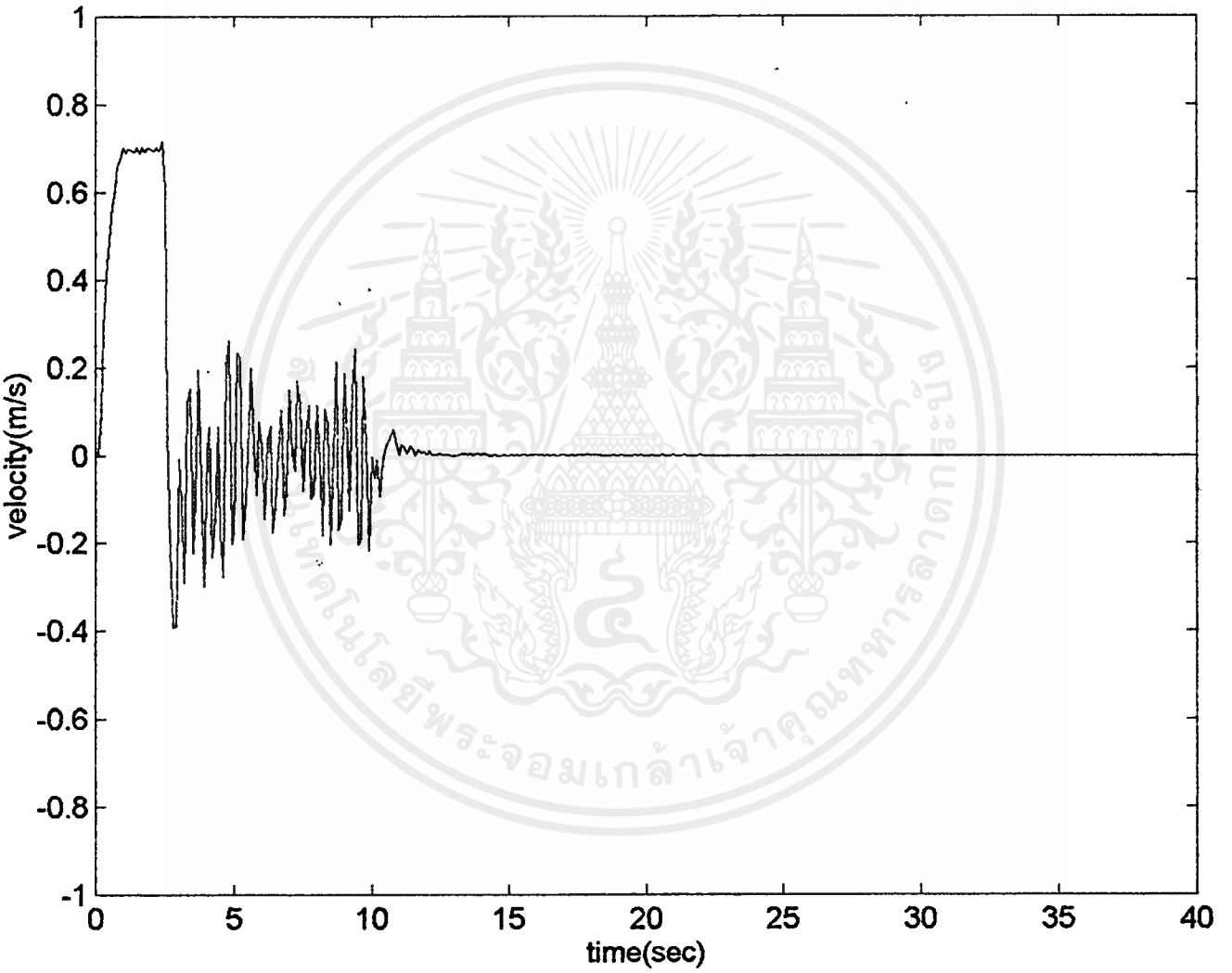

รูปที่ 5.3ข รูปผลตอบสนองมุมในการแกว่ง



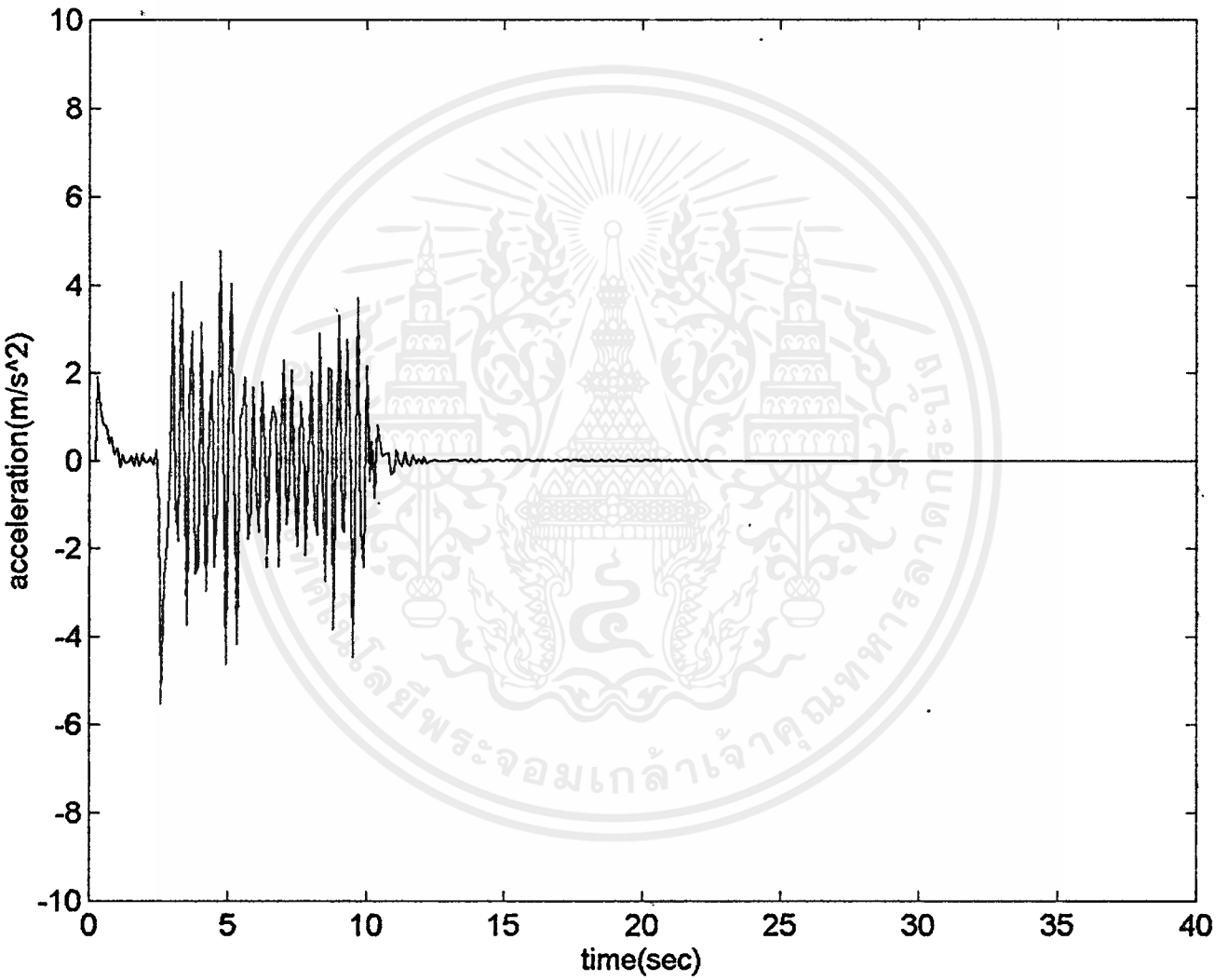

รูปที่ 5.3ค รูปผลตอบสนองของความเร็วเชิงมุม




รูปที่ 5.3ง รูปผลตอบสนองของตำแหน่งของรถเครน

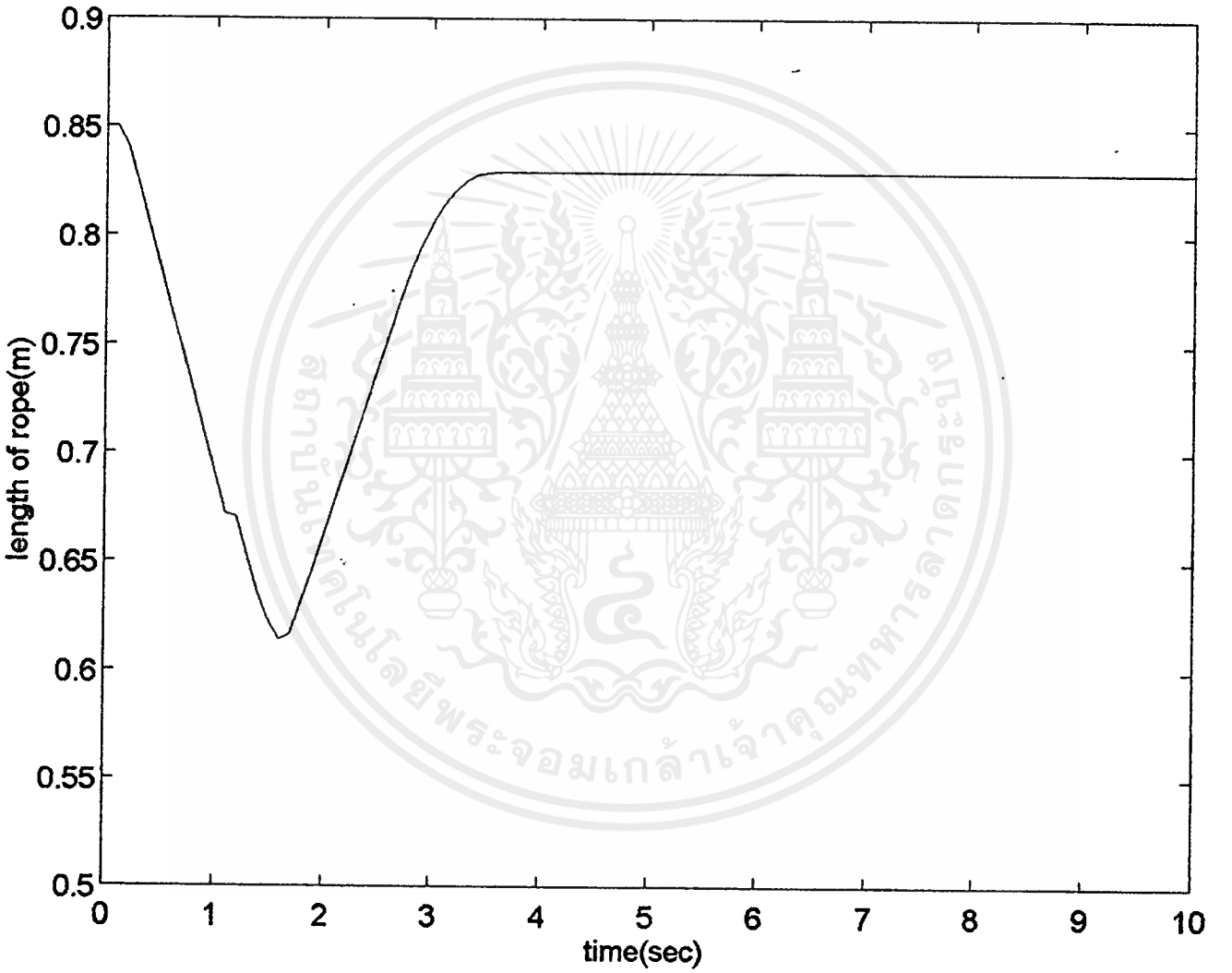



รูปที่ 5.3จ รูปผลตอบสนองของความเร็ว

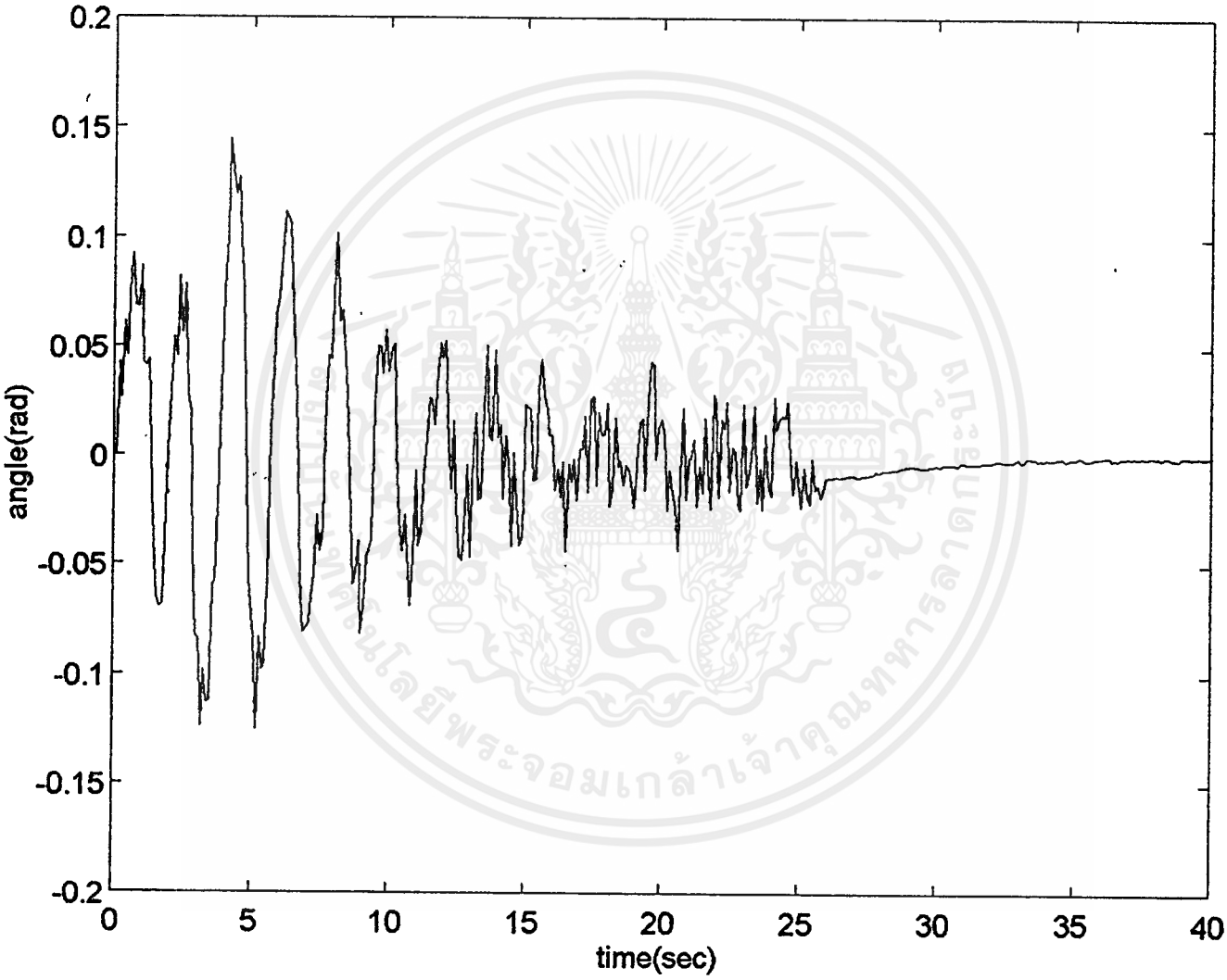



รูป 5.3ฉ รูปผลตอบสนองของความเร่ง

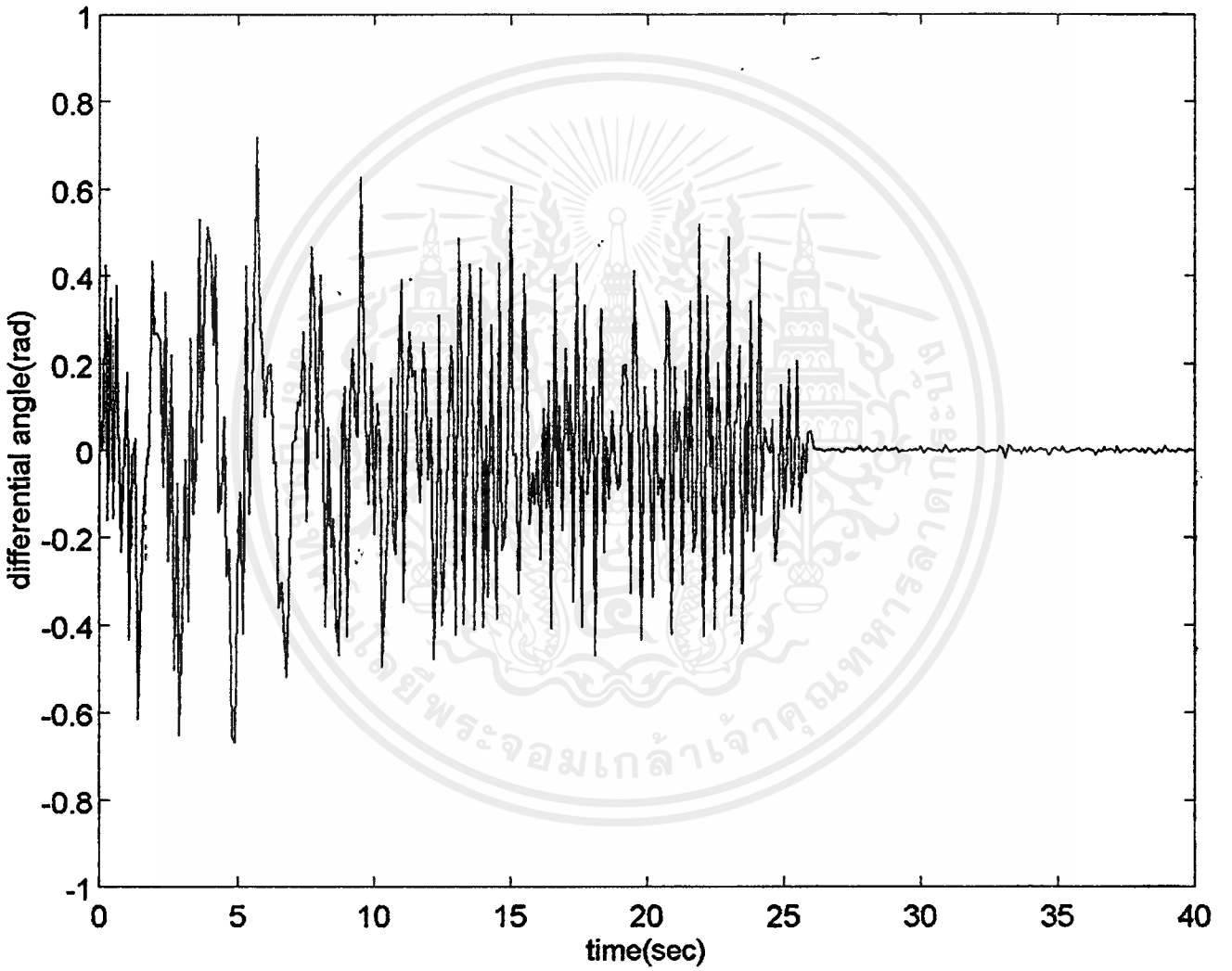
5.2.2 ผลการทดลองการเคลื่อนที่ของเครน เมื่อให้ความยาวเชือกของภาระเปลี่ยนแปลง



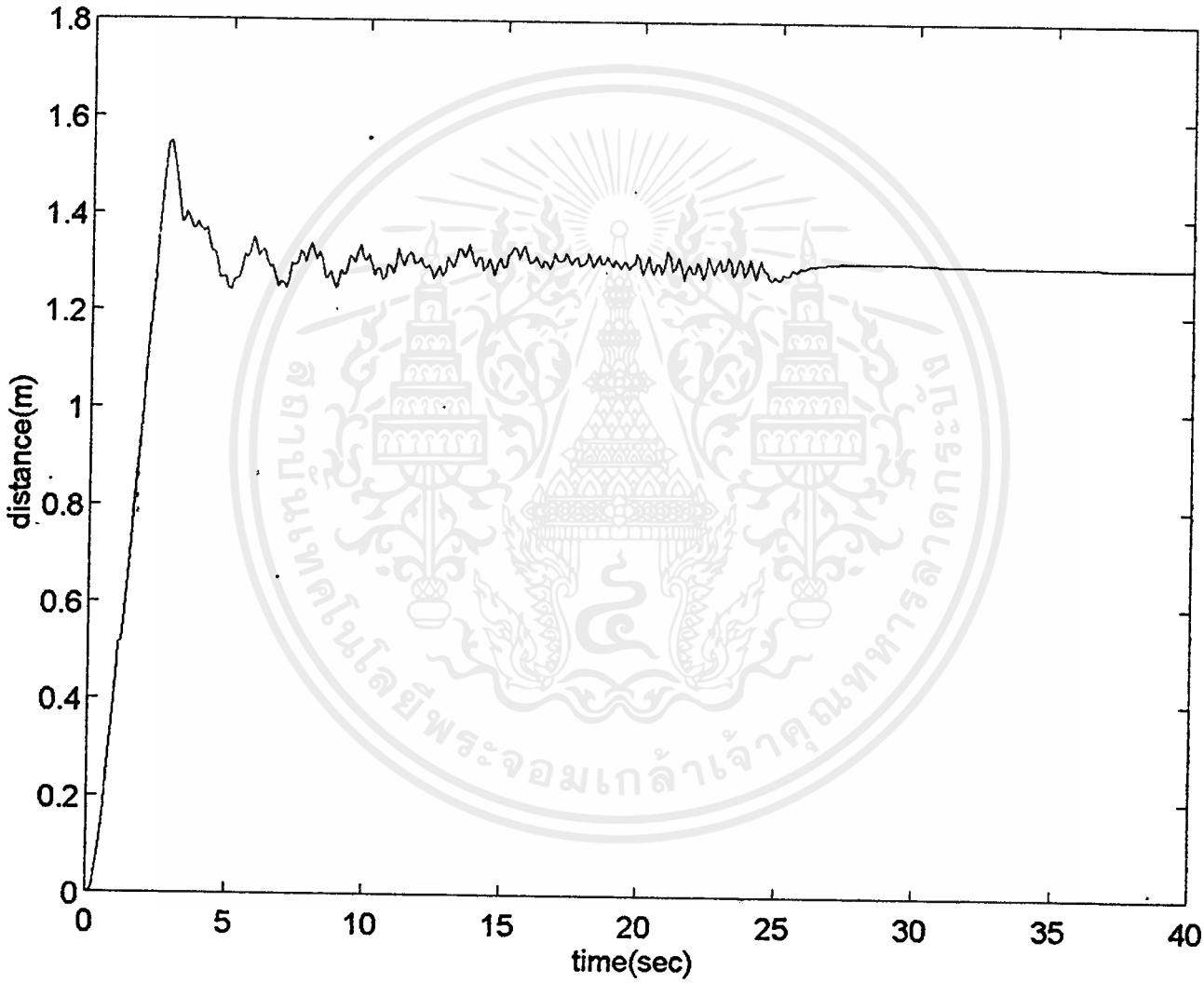

รูปที่ 5.4ก รูปแสดงความยาวเชือกที่เปลี่ยนไปต่อเวลาที่ใช้ในการเคลื่อนที่



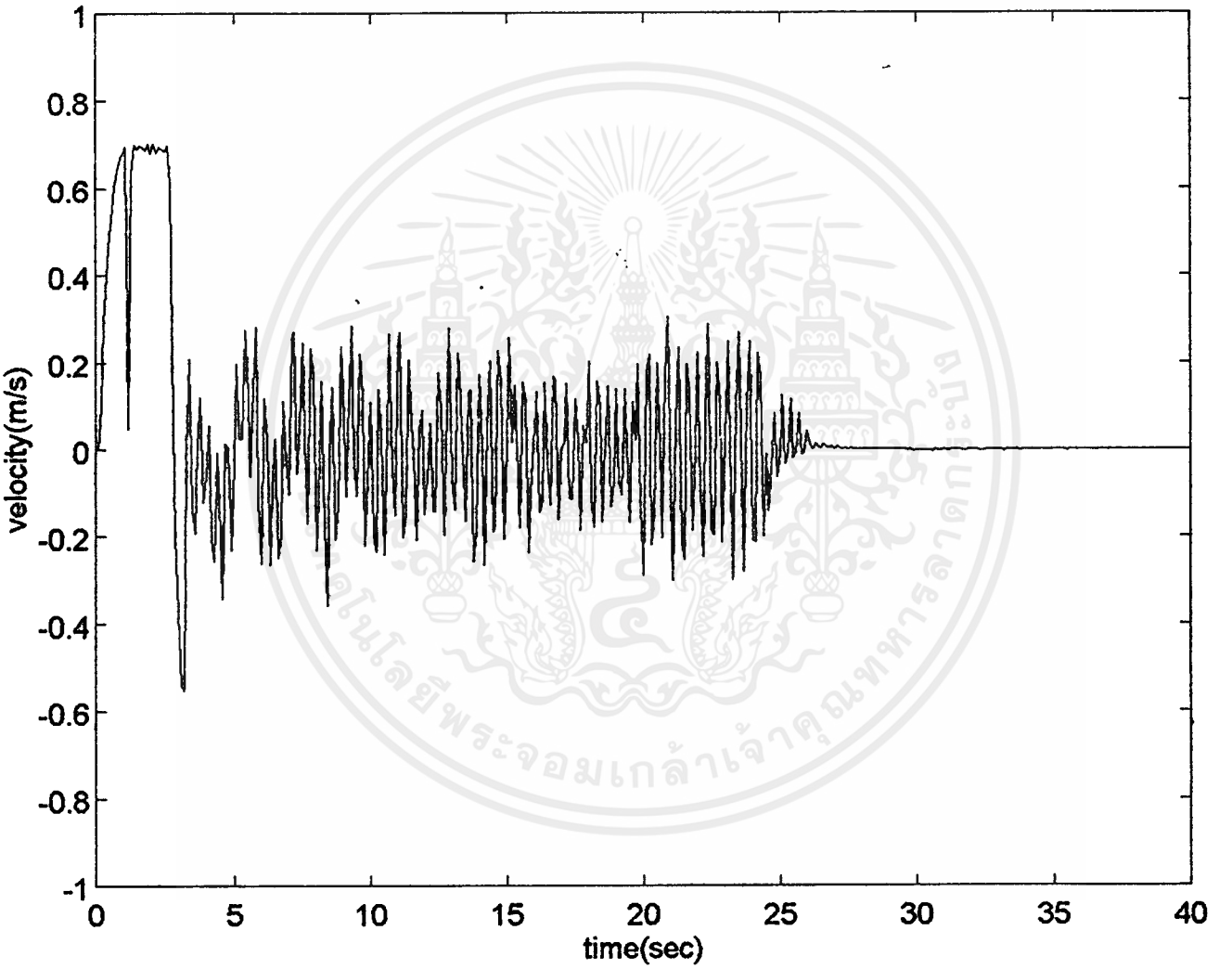

รูปที่ 5.4ข รูปผลตอบสนองมุมในการแกว่ง



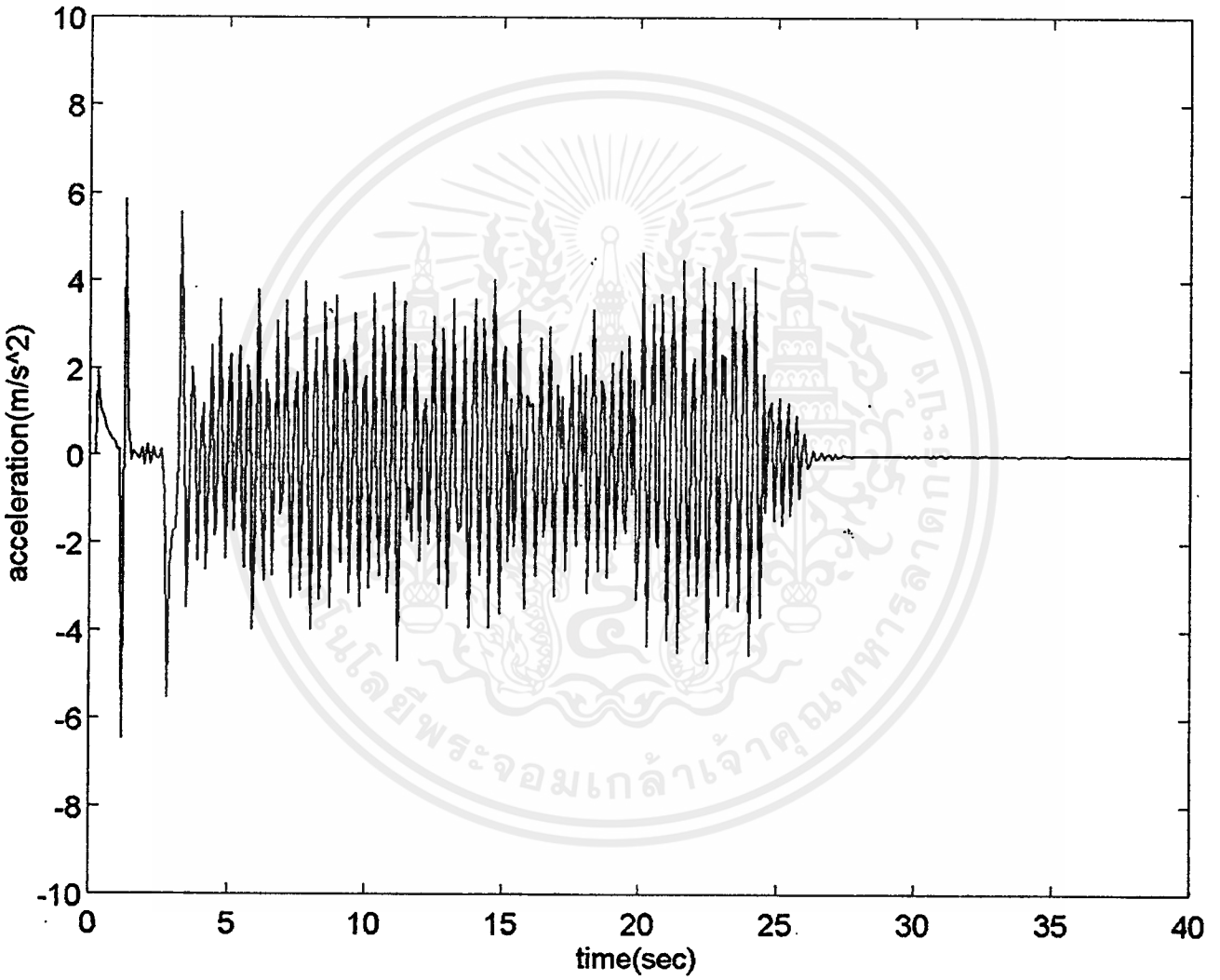

รูปที่ 5.4ค รูปผลตอบสนองของความเร็วเชิงมุม




รูปที่ 5.4ง รูปผลตอบสนองของตำแหน่งของรถเครน




รูปที่ 5.4จ รูปผลตอบสนองของความเร็ว




รูป 5.4ฉ รูปผลตอบสนองของความเร่ง

# บทที่6

## ผลสรุปและข้อเสนอแนะของระบบควบคุมการเคลื่อนที่ของเครน

### 6.1 สรุปผลการทดลอง

1.จากการทดลองการควบคุมการเคลื่อนที่ของเครนซึ่งมีตัวควบคุมที่ได้ถูกออกแบบโดยกำหนดให้ R=200 และ Q= diag([1000 1 1000 1 1]) เมื่อกำหนดให้รถเครนเคลื่อนที่ไปเป็นระยะทาง1.3 เมตร และ มีความยาวเชือกคงที่ (ในกรณีแรก) และมีความยาวเชือกเปลี่ยนแปลง(ในกรณีที่2) เพื่อหลบหลีกสิ่งกีดขวาง พบว่า ระบบในกรณีแรก จะเข้าสู่ สถานะคงตัวได้เร็วกว่าระบบในกรณีที่ 2 ในขณะที่มีมุมของการแกว่งที่น้อยกว่าโดยจะมีมุมในการแกว่งสูงสุดที่ 0.09 rad สำหรับระบบในกรณีที่2นั้น จะมีการแกว่งสูงสุดที่ 0.14 rad ในขณะที่ rise time จะมีค่าใกล้เคียงกัน คือประมาณ 2.5 วินาที โดยมี maximum overshoot ของระบบในกรณีที่ 1 ประมาณ 15 % และระบบในกรณีที่ 2 ประมาณ 19 %

### 6.2 ข้อเสนอแนะ

1.ในการออกแบบระบบควบคุมแบบ LQR นั้น ลักษณะของผลตอบสนองที่ได้จะขึ้นอยู่กับความต้องการของผู้ออกแบบว่า ต้องการให้ผลตอบสนองของแต่ละสเตทเป็นอย่างไร ซึ่งผู้ออกแบบควรลองเปลี่ยนค่า Q และ R หลาย ๆค่า เพื่อที่จะหาค่าที่เหมาะสมที่สุดของระบบควบคุมนั้น ๆ

2.ควรพัฒนาตัวควบคุมแบบต่างๆ มาทดลองใช้กับระบบ เพื่อการศึกษาและทำความเข้าใจในทฤษฎีของระบบควบคุมต่อไป

3. การใช้รีเลย์มาควบคุมมอเตอร์นั้น จะควบคุมให้การเปลี่ยนแปลงของความยาวเชือกมีค่าถูกต้องนั้น จะทำได้ยากถ้านำวงจรขับมอเตอร์มาควบคุมแทนที่ วงจรรีเลย์ จะได้ความถูกต้องกว่า.

# ภาคผนวก ก. การใช้งานการ์ดอินเตอร์เฟส PCL 714

00714-90001 March, 1988 Rev. 2B

Table of Contents

## 1.1. General Description

The PCL-714 Super-Lab card is a high performance analog/digital I/O card that offers 5 most desirable measurement and control functions on a single board.

Its versatile functions include 16 differential analog inputs, 2 analog outputs (one is optional), 16 digital inputs, 16 digital outputs and a programmable interval timer.

Designed with engineering advancement and user satisfaction in mind, this full size card specially offers 14 bit resolution for both D/A and A/D conversion and turns the IBM PC* into a high precision voltage measurement and signal analysis instrument. In addition to its highly condensed features, the versatility of the card can be further enhanced with the use of optional daughter cards such as PCLD-780, PCLD-782 and PCLD-785.

The PCLD-785 is a 16-channel relay output card which can be driven by the digital output of the PCL-714 card. The PCLD-782 is a 16-channel opto-isolated digital input card which provides an easy way to input digital data to the PCL-714. The PCLD-780 wiring terminal board allows analog and/or digital I/O's be easily connected to the PCL-714. All three daughter boards can be connected directly to the PCL-714 via 20 pin flat ribbon cables.

As a further convenience in your application, the PCL-714 card is also supported by the PC-LabDAS, a menu driven data acquisition software and the UnkelScope, a powerful waveform analysis software.

Together with all the features and supports, the PCL-714 is intended to provide a total and cost effective solution for your application need.

* IBM PC is a trademark of International Business Machines Corporation.

## 1.2. Features, Applications and Specifications

Features

- 16 differential analog input with 14 bit resolution
- 2 analog output with 14 bit resolution
- Hardware successive approximation for faster conversion time and higher throughput
- A/D conversion time less than 40 microseconds
- 16 digital input channels
- 16 digital output channels
- All digital input and output channels are

TTL compatible
. Programmable interval timer
. Programmable pacer trigger

Applications

. Industrial measurements/automation
. Laboratory measurements/automation
. Signal analysis
. Process control/monitoring
. Contact closure monitoring
. Switch panel status sense
. Industrial on/off control
. BCD interface
. Digital I/O control
. Period and pulse width measurement
. Event and frequency counting

Specifications

Analog to Digital
Input Range : -5V to +5V
Input Channels : 16 differential
Accuracy : 0.15% max. at 25 deg.C.
Input Impedance: > 10 mega Ohms
Conversion Time: < 40 microseconds
Resolution : 14 bits

Digital to Analog
Output Range : Bipolar -5V to +5V
Output Channels: 1 standard, 1 optional
Accuracy : 0.1% max. at 25 deg.C.
Settling time : < 30 microsecond for 5V step
Resolution : 14 bits

Digital Input
Input Low Level : Min. -0.5V, max. 0.8V

Input High Level: Min. 2.0V, max. 5.0V
Input Loading : 0.2 mA at 0.4V
Input Hysteresis: Typical 0.4V, min. 0.2V

Digital Output
Output Low Level : Max. 0.5V at 24 mA
(sink) Max. 0.4V at 12 mA
Output High Level: Min. 2.0V at 15 mA
(source) Min. 2.4V at 3 mA
Driving Capacity : 15 TTL's at least

Power Consumption : < 800 mA at 5V
< 50 mA at +12V
< 50 mA at -12V

General
Dimensions : 13 3/8" x 3 3/4"
34 cm x 9.5 cm
Bus : IBM PC bus
Slot : One 62-pin slot
I/O Port Base Address :
Hex 200 - hex 3F0
I/O Port Space Occupied: 12

+----------+
| CAUTION |
+----------+

The PCL-714 Super-Lab Card uses 4052 CMOS chips as multiplexer for analog input channels. Do not apply any voltage higher than +12.5V or lower than -5.5V. Out of limit input voltage may cause permanent damage to the multiplexer circuits.

# 2. INSTALLATION

## 2.1. Initial Inspection

Inside the shipping container, you should find this operating manual and the PCL-714 card.

The PCL-714 was carefully inspected both mechanically and electrically before shipment. It should be free of marks and scratches and in perfect electrical order on receipt.

Remove the PCL-714 interface card from its protective packaging by grasping the metal rear panel. Keep the anti vibration package since it may be used to return the card if it needs repair. The package may also be used if the card is stored outside of the computer.

The board should be handled only by the edges. The integrated circuits on the board can be damaged by static electric discharge.

## 2.2. Base Address Selection

Most of the peripheral devices and the interface adapters in the PC are controlled and sensed using the digital input and output ports. These ports are addressed using the I/O port address space of the 8088 or 80286 microcomputer.

The I/O port base address for the PCL-714 is selectable by a 5 position DIP switch. Valid addresses are from hex 200 to hex 3F0. Refer to Figure 2.1 for the location of the DIP switch (SW1).

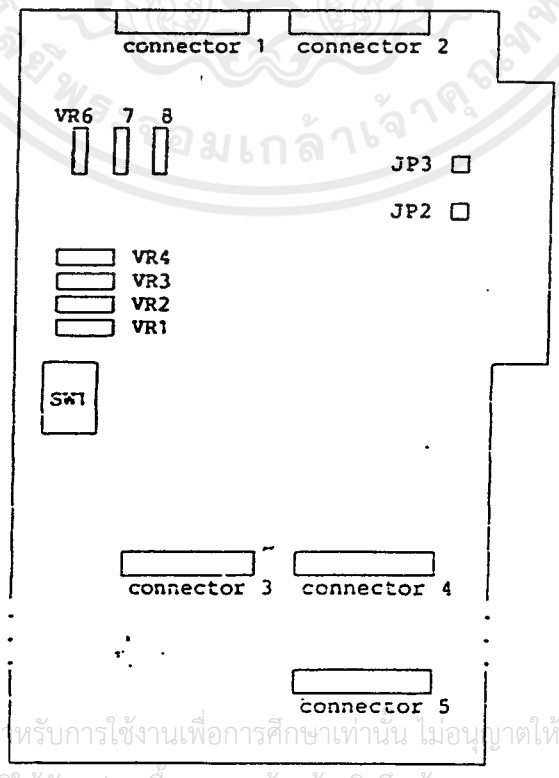



Figure 2.1 The PCL-714 Card

The required switch settings for various base addresses are illustrated as below:

Note: - ON = 0 , OFF = 1
- "X" means "don't care"
- 1..5 are switch positions
- A4..A8 correspond to address lines of the PC bus. A9 is hard wired to be 1.
- * means factory setting

| I/O port address (Hex) | switch position 1 A8 | 2 A7 | 3 A6 | 4 A5 | 5 A4 |
|---|---|---|---|---|---|
| 200-20F | 0 | 0 | 0 | 0 | 0 |
| . | | | | | |
| . | | | | | |
| . | | | | | |
| *220-22F | 0 | 0 | 0 | 1 | 0 |
| . | | | | | |
| . | | | | | |
| . | | | | | |
| 2E0-2EF | 0 | 1 | 1 | 1 | 0 |
| 2F0-2FF | 0 | 1 | 1 | 1 | 1 |
| 3E0-3EF | 1 | 1 | 1 | 1 | 0 |
| 3F0-3FF | 1 | 1 | 1 | 1 | 1 |

## 2.3. Connector Pin Assignments

The PCL-714 card is equipped with two 20-pin insulation displacement (mass termination) connectors accessible from the rear plate and three other 20-pin insulation displacement connectors on board. All these connectors can be connected to flat cables of the same type. Please refer to Fig. 2.1 for the location of each connector.

The following diagrams below show their pin assignments.

Legend:
- A/D H - differential analog input high
- A/D L - differential analog input low
- D/A - analog output
- D/O - digital output
- D/I - digital input
- GND - ground
- CLK - the clock input for the 8253
- GATE - the gate input for the 8253
- OUT - the signal output of the 8253

Connector 1:

| | | | | | | |
|---|---|---|---|---|---|---|
| A/D H | 0 | 1 | 2 | A/D L | 0 |
| A/D H | 1 | 3 | 4 | A/D L | 1 |
| A/D H | 2 | 5 | 6 | A/D L | 2 |
| A/D H | 3 | 7 | 8 | A/D L | 3 |
| A/D H | 4 | 9 | 10 | A/D L | 4 |
| A/D H | 5 | 11 | 12 | A/D L | 5 |
| A/D H | 6 | 13 | 14 | A/D L | 6 |
| A/D H | 7 | 15 | 16 | A/D L | 7 |
| A/D H | 8 | 17 | 18 | A/D L | 8 |
| A/D H | 9 | 19 | 20 | A/D L | 9 |

Connector 2:

| | | | |
|---|---|---|---|
| A/D H 10 | 1 | 2 | A/D L 10 |
| A/D H 11 | 3 | 4 | A/D L 11 |
| A/D H 12 | 5 | 6 | A/D L 12 |
| A/D H 13 | 7 | 8 | A/D L 13 |
| A/D H 14 | 9 | 10 | A/D L 14 |
| A/D H 15 | 11 | 12 | A/D L 15 |
| D/A 1 | 13 | 14 | GND |
| D/A 2 | 15 | 16 | GND |
| | 17 | 18 | GND |
| +5V | 19 | 20 | +12V |

Connector 3:

| | | | |
|---|---|---|---|
| D/O 0 | 1 | 2 | D/O 1 |
| D/O 2 | 3 | 4 | D/O 3 |
| D/O 4 | 5 | 6 | D/O 5 |
| D/O 6 | 7 | 8 | D/O 7 |
| D/O 8 | 9 | 10 | D/O 9 |
| D/O 10 | 11 | 12 | D/O 11 |
| D/O 12 | 13 | 14 | D/O 13 |
| D/O 14 | 15 | 16 | D/O 15 |
| GND | 17 | 18 | GND |
| +5V | 19 | 20 | +12V |

Connector 4:

| | | | |
|---|---|---|---|
| D/I 0 | 1 | 2 | D/I 1 |
| D/I 2 | 3 | 4 | D/I 3 |
| D/I 4 | 5 | 6 | D/I 5 |
| D/I 6 | 7 | 8 | D/I 7 |
| D/I 8 | 9 | 10 | D/I 9 |
| D/I 10 | 11 | 12 | D/I 11 |
| D/I 12 | 13 | 14 | D/I 13 |
| D/I 14 | 15 | 16 | D/I 15 |
| GND | 17 | 18 | GND |
| +5V | 19 | 20 | +12V |

Connector 5:

| | | | |
|---|---|---|---|
| | 1 | 2 | |
| | 3 | 4 | |
| | 5 | 6 | GATE 1 |
| | 7 | 8 | CLK 0 |
| | 9 | 10 | OUT 0 |
| | 11 | 12 | GATE 0 |
| | 13 | 14 | |
| | 15 | 16 | |
| GND | 17 | 18 | GND |
| +5V | 19 | 20 | |

2.4. DMA Acknowledge and Request

When DMA operation is needed on the PCL-714, JP2 and JP3 are used to select the proper acknowledge signal (1-3) and request signal (1-3).

```
JP2  . . .    JP3  . . .
     . . .         . . .
     1 2 3         1 2 3

     DACK          DRQ
```

It is recommended that acknowledge signal 1 (DACK1) and request signal 1 (DRQ1) be used on the PCL-714. Please note that always select signals on the same level, for example, if DACK2 is selected, then DRQ2 must be selected.

Refer to the data sheet of the 8237 DMA controller for more detailed information on DMA operation.

## 2.5. Installing The PCL-714

POWER MUST ALWAYS BE SWITCHED OFF when removing or inserting the PCL-714 card and connecting or disconnecting cables.

Use a screw driver to remove the cover mounting screws from the rear of the system unit.

Slide the system unit's cover away from the rear and to the front. When the cover will go no further, tilt it up, remove it from the base, and set it aside.

The PCL-714 is configured at the factory for the IBM PC, PC/XT, PC/AT and all IBM PC compatibles. If you need further changes to the configuration please refer to Chapter 2. Installation.

Use a screw driver to remove the screw that secures the expansion slot cover. Save the screw for installation of the interface card.

The two rear connectors should be pressed through the rear panel first, then press the card carefully into the main board expansion slot.

Secure the PCL-714 with the 3/16" mounting screw, then attach an appropriate cable to the connector.

Slide the system unit's cover back on. Align the system unit tabs with the cover holes and reinstall the 1/4" mounting screws.

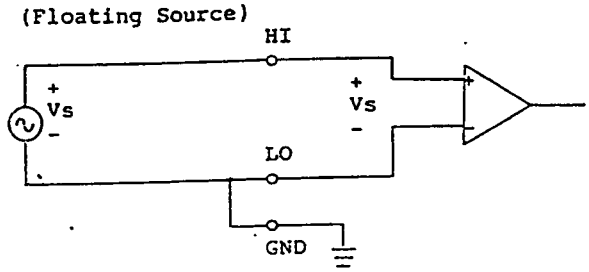
## 2.6. Signal Connection

### 2.6.1. Analog Signal Connection

PCL-714 has 16 channel differential analog inputs. It can measure 16 signal sources by scanning the channels. The differential input responds only to difference signals between the High and Low inputs.
If the signal source has no connection to ground, it is called "floating source". To measure a floating source, the PCL-714 channel should be connected as:

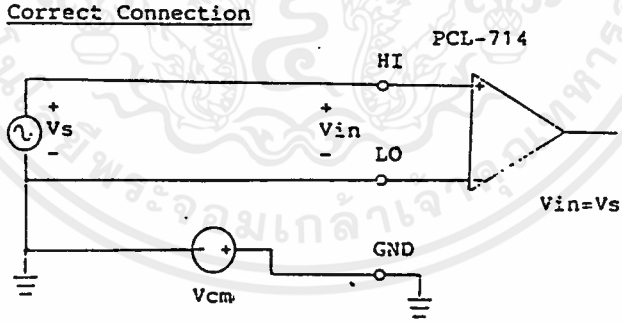
PCL-714

(Floating Source)

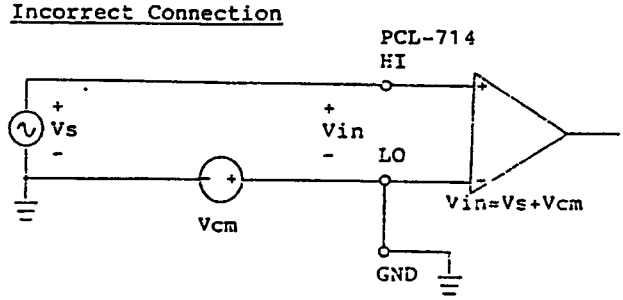



A connection must exist between LO and GND to define Common input voltage.
If the signal source has one side connected to a local ground. The signal source ground and the PCL-714 ground will not be at the same voltage as they are connected through the ground return of the equipment and building wiring. The difference between the ground voltages forms a common mode voltage. To avoid the ground loop noise effect, the signal ground should be connected to PCL-714 LO input. The LO input should not be connected to PCL-714 GND directly. For better grounding, in some cases, a wire connection between the PCL-714 GND and signal source is necessary.

Correct Connection

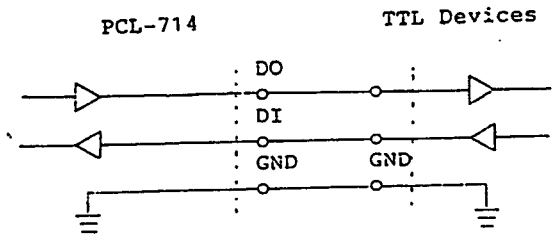



Incorrect Connection

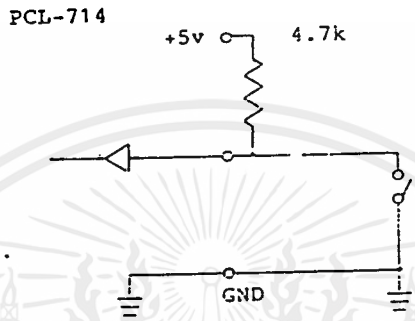



2.6.2. Digital Signal Connection

The PCL-714 has 16 digital input and 16 digital output channels. The digital I/O levels are TTL compatible. To transmit or receive digital signals to/form other TTL devices, the connection is:




To receive an OPEN/SHORT signal from switch or relay, a pull up resistor must be added to ensure the nigh level when open.




## 3. REGISTER STRUCTURE AND FORMAT

The most important issue in programming the PCL-714 is understanding the meaning of the 12 registers addressable from the selected I/O port base address.

The following diagram shows the relative location of each register as to the base address and its usage.

Legend:
- LSB - least significant byte
- MSB - most significant byte
- R - read operation on that byte
- W - write operation on that byte

| | | | |
|---|---|---|---|
| BASE+0 | R | LSB or MSB of Counter 0 | 8253 timer. |
| | W | LSB or MSB of Counter 0 | |
| BASE+1 | R | LSB or MSB of Counter 1 | |
| | W | LSB or MSB of Counter 1 | Ch.1,2 |
| BASE+2 | R | LSB or MSB of Counter 2 | used as |
| | W | LSB or MSB of Counter 2 | pacer |
| BASE+3 | W | CONTROL BYTE | |
| BASE+4 | R | A/D low byte | |
| | W | CH1 D/A low byte | |
| BASE+5 | R | A/D high byte (bit 0-5)<br>bit 6 =1, A/D conversion not ready<br>bit 6 =0, A/D conversion ready | |
| | W | CH1 D/A high byte (bit 0-5) | |
| BASE+6 | R | D/I channel 0-7 | |
| | W | CH2 D/A low byte | |
| BASE+7 | R | D/I channel 8-15 | |

| | | |
|---|---|---|
| BASE+8 | | Not used |
| BASE+9 | | Not used |
| BASE+10 | W | Bit 0-3 are used to select the desired A/D channel |
| BASE+11 | W | A/D trigger mode control |
| BASE+12 | | Not used |
| BASE+13 | W | D/O channel 0-7 |
| BASE+14 | W | D/O channel 8-15 |

## 4. A/D CONVERSION

### 4.1. Data Format

In order to perform the A/D conversion, you have to be familiar with the usage of the following four registers:

BASE+4 :
This byte, together with bit 0-5 of the byte at BASE+5, form a 14 bit number which is the result of the A/D conversion. The number can be from 0 to 16383.

BASE+5 :
The use of bit 0-5 is described as above. Bit 6 reflects the status of the conversion; 0 = ready, 1 = not ready.

BASE+10
Bit 0-3 of this register are used to select the desired A/D channel on which the A/D conversion will be performed. Bit 4-7 are ignored.

BASE+11
Trigger Mode Control. This register is used to make a direct trigger or to enable or disable pacer trigger.

Bit 0 : 0 No direct trigger
1 Make a direct trigger
Bit 1 : 0 Disable pacer trigger
1 Enable pacer trigger

## 4.2. Trigger Mode

### 4.2.1. Direct Trigger Mode

When writing a byte with bit 0 setting 1 to the register BASE+11, the A/D converter is triggered directly. A reading will be ready after A/D conversion is completed.

The following programming examples written in BASIC is to sample all the 16 A/D channels and print the results.

```
10  BASE%=&H220  'Base address
20  FOR CH%=0 TO 15
30  GOSUB 70
40  PRINT CH%,V  'Print channel no. and value
50  NEXT CH%
60  END
70  '****** A/D routine ******
80  OUT BASE%+10,CH%  'Select the channel
90  OUT BASE%+11,1    'Trigger the conversion
100 'Loop if the conversion is not complete
110 HI=INP(BASE%+5):IF HI>=64 THEN 110
120 'Read the A/D low byte
130 LO=INP(BASE%+4)
140 'Convert the value to voltage,
150 ' subtracted by 8192, because the input
160 ' is bipolar, +5V to -5V
170 V=(HI*256+LO-8192)*10/16384
180 RETURN
```

### 4.2.2. Pacer Trigger Mode

The PCL-714 has a Intel 8253 chip worked as a timer/counter. The counter channel 1 and 2 of 8253 chip are configured to be a pacer to offer A/D converter trigger pulses with precise period in pacer trigger mode. To enable the pacer trigger, write a byte with bit 1 setting 1 to the register BASE+11. The following program section is used to set the sampling period:

```
OUT BASE+3,&H74
OUT BASE+1,LB1
OUT BASE+1,HB1
OUT BASE+3,&HB4
OUT BASE+2,LB2
OUT BASE+2,HB2
OUT BASE+11,2
```

The sampling period is

( HB1 *256 + LB1 ) * ( HB2 * 256 + LB2 ) * 0.5 microseconds

For example, if LB1=1, HB1=0, LB2=200 and HB2=0, then the sampling period is 100 microseconds.

The GATE1 input of connector CN4 can be used to disable the pacer by inserting it low. This allows the pacer be gated by external signal.

## 5. D/A CONVERSION

Programming the D/A channels is easier than programming the A/D channels. All you have to do is writing the proper value of the desired output voltage into the registers.

The following programming examples in BASIC show how to program D/A channel 1 to generate a voltage of +3V.

```
10 'Figure out the value for the low and
20 ' high byte
30 V=3.0
40 X%= (V/10*16384)+8192
50 HI%= X%\256
60 LO%= X%-(HI%*256)
70 'Write the value into registers
80 OUT &H220+4,LO% : OUT &H220+5,HI%
90 END
```

## 6. DIGITAL I/O

On the PCL-714 card, 16 digital input channels and 16 digital output channels are provided. Four I/O ports are reserved for accessing these channels.

The four ports are allocated as:

| | | |
|---|---|---|
| BASE+6 | D/I channel | 0 - 7 |
| BASE+7 | D/I channel | 8 - 15 |
| BASE+13 | D/O channel | 0 - 7 |
| BASE+14 | D/O channel | 8 - 15 |

A reading operation on any of the D/I ports will read in the value of the 8 corresponding digital input channels. To access the D/I ports in BASIC, use the following statement :

```
VALUE = INP(ADDRESS)
```

Where ADDRESS is BASE+6 or BASE+7.

A writing operation to any of the D/O ports will set the desired value on the 8 corresponding digital output channels. To access the D/O ports in BASIC, use the statement shown below:

```
OUT ADDRESS,VALUE
```

Where ADDRESS is BASE+13 or BASE+14.

# 7. PROGRAMMABLE INTERVAL TIMER

## 7.1. The Intel 8253

The Intel 8253 Programmable interval timer is used on the PCL-714 card. It is organized as 3 independent 16-bit counters, each with a count rate of up to 2.6 MHz. Each counter can be programmed to have a count from 0 up to 65,535. Each counter can also be set to operate in one of the 6 different modes of operation. All modes of operation are software programmable.

The main counter/timer functions that can be implemented with the 8253 are:

. Programmable Rate Generator
. Event Counter
. Real Time Clock
. Digital One-Shot
. Time Delay Generator
. Square Wave Generator

For more information regarding the 8253 programmable interval timer, please refer to the Intel Microsystem Components Handbook (Microprocessors and Peripherals Volume II).

## 7.2. The Control Byte

The 8253 programmable interval counter occupies 4 I/O address locations in the PCL-714 I/O address map as mentioned in Section 3.1. Their addresses are :

| Address | Register Type | Description |
|---|---|---|
| BASE+0 | read/write | Counter 0 |
| BASE+1 | read/write | Counter 1 |
| BASE+2 | read/write | Counter 2 |
| BASE+3 | write only | Control Byte |

Before loading or reading any of the individual counters, the 8253 control byte must be loaded with data setting the counter selected, the operating mode, the type of read or write operation that will be performed, and the modulus (binary or BCD).

The format of the Control Byte is:

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | SC1 | SC0 | RL1 | RL0 | M2 | M1 | M0 | BCD |

SC1-0 : Select counter.

| SC1 | SC0 | Counter |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 2 |
| 1 | 1 | illegal |

RL1-0 : Select the read or load operation

| RL1 | RL0 | Operation |
|---|---|---|
| 0 | 0 | Counter latch |
| 0 | 1 | read/load LSB |
| 1 | 0 | read/load MSB |
| 1 | 1 | read/load LSB first, then MSB. |

The Counter latch command is used to read back the value of a Counter without disturbing the count in progress.

M2-0 : Select the operating mode

| M2 | M1 | M0 | Mode |
|---|---|---|---|
| -- | -- | -- | ---- |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| X | 1 | 0 | 2 |
| X | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |

See Section 3.3.3 for the definition of each mode.

BCD : Selects binary or BCD counting

| BCD | Type |
|---|---|
| --- | ---- |
| 0 | binary counter 16-bits |
| 1 | BCD counter |

If the modulus is set to be binary, the count can be any number from 0 up to 65,535. If the modulus is set to be BCD (binary coded decimal), then the count can be set as any number from 0 to 9,999.

## 7.3. Mode Definitions

The definitions of the six operating modes are described here. Refer to the data sheet of the 8253 for more details.

Mode 0: Interrupt On Terminal Count

The output will be initially low after the mode set operation. After the count is loaded into the selected count register, the output will remain low and the counter will count. When terminal count is reached, the output will go high and remain high until the selected counter is reloaded with the mode or a new count is loaded. The counter continues to decrement after terminal count has been reached.

Rewriting a counter register during counting results in the following:

1. Write 1st byte stops the current counting.
2. Write 2nd byte starts the new count.

Mode 1: Programmable One-Shot

The output will go low on the count following the rising edge of the gate input. The output will go high on the terminal count. If a new count value is loaded while the output is low, it will not affect the duration of the one-shot pulse until the succeeding trigger. The current count can be read at any time without affecting the one-shot pulse.

The one-shot is retriggerable, hence the output will remain low for the full count after any rising edge of the gate input.

Mode 2: Rate Generator

Divide by N counter. The output will be low

for one period of the input clock. The period from one output pulse to the next equals the number of input counts in the counter register. If the count register is reloaded between output pulses, the present period will not be affected, but the subsequent period will reflect the new value.

The gate input, when low, will force the output high. When the gate input goes high, the counter will start from the initial count. Thus, the gate input can be used to synchronize the counter.

When this mode is set, the output will remain high until after the count register is loaded. The output then can also be synchronized by software.

Mode 3: Square Wave Rate Generator

Similar to mode 2, except that the output will remain high until one half the count has been completed (for even numbers) and go low for the other half of the count. This is accomplished by decrementing the counter by two on the falling edge of each clock pulse. When the counter reaches terminal count, the state of the output is changed and the counter is reloaded with the full count and the whole process is repeated.

If the count is odd and the output is high, the first clock pulse (after the count is loaded) decrements the count by 1. Subsequent clock pulses decrement the count by 2. After timeout, the output goes low and the full count is reloaded. The first clock pulse (following the reload) decrements the counter by 3. Subsequent clock pulses decrement the count by two until timeout. Then the whole process is repeated. In this way, if the count is odd, the output will be high for (N+1)/2 counts and low for (N-1)/2 counts.

Mode 4: Software Triggered Strobe

After the mode is set, the output will be high. When the count is loaded, the counter will begin counting. On terminal count, the output will go low for one input clock period, then will go high again.

If the count register is reloaded during counting, the new count will be loaded on the next CLK pulse. The count will be inhibited while the GATE input is low.

Mode 5: Hardware Triggered Strobe

The counter will start counting after the rising edge of the trigger input and will go low for one clock period when the terminal count is reached. The counter is retriggerable. The output will not go low until the full count after the rising edge of any trigger.

7.4. Loading and Reading The Counters

The programming procedure for the 8253 is

1. For each Counter, The Control Byte must be written before the initial count is written.
2. The initial count must follow the count format specified in the Control Byte (LSB only, MSB only or LSB first then MSB).

Since the Control Byte register and the three counters have separate addresses and each Control Byte specifies the Counter it applies to (by SC1 and SC0), no special instruction sequence is required. Any programming sequence that follows the conventions above is acceptable.

A new initial count may be written to a Counter at any time without affecting the Counter's programmed mode in any way. Counting will be affected as described in the mode definitions. The new count must follow the programmed count format.

If a Counter is programmed to read/load two byte counts, the following precaution applies: A program must not transfer control between writing the first and second byte to another routine which also writes into that same Counter. Otherwise, the Counter will be loaded with an incorrect count.

Some programming examples in BASIC:

```
10  'Set base address
20  BASE%=&H220
30  'Control byte for Counter 0:
40  'Write LSB only, mode=3, BCD=0
50  OUT BASE%+3,&H16
60  'Output LSB=10
70  LSB%=10
80  OUT BASE%,LSB%
90  'Control bytes for Counter 1 and 2
100 'Write LSB first, then MSB, mode=1, BCD=0
110 OUT BASE%+3,&H72    'For Counter 1
120 OUT BASE%+3,&HB2    'For Counter 2
130 'X% is the count for both counters
140 X%=1000
150 MSB%=INP(X%/256)    'Figure out the MSB
160 LSB%=X%-(256*MSB%) 'Figure out the LSB
170 OUT BASE%+1,LSB%    'Load Counter 1 LSB
180 OUT BASE%+1,MSB%    '               MSB
190 OUT BASE%+2,LSB%    'Load Counter 2 LSB
200 OUT BASE%+2,MSB%    '               MSB
210 END
```

It is often desirable to read the value of a Counter without disturbing the count in progress. Usually, the method used is called the Counter Latch Command method which allows the user to read the latched count value of the selected Counter.

The command is written into the Control Byte Register and has the format shown below. The command applies to the Counter selected by the SC1 and SC2 bit in the Control Byte. And the command distinguishes itself from other commands by setting bit 4 and 5 to 0.

| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | SC1 | SC0 | 0 | 0 | X | X | X | X |

SC1-0: Select the desired Counter
X : Means don't care

The instruction sequence is :

1. Load the Control Byte
2. Read the less significant byte
3. Read the most significant byte

Some programming examples in BASIC:

```
10  PORT%=&H220  'Counter 0 address
20  'Load LSB then MSB, mode=0, BCD=0
30  OUT PORT%+3,&H30 'Control byte
40  L%=0: H%=32: CNT%=H%*256+L%
50  OUT PORT%,L% 'Load LSB
60  OUT PORT%,H% 'Load MSB
70  T1=TIMER 'Get current time
80  'Control byte for read the count
90  OUT PORT%+3,0
100 'Read the LSB and MSB
110 C= INP(PORT%)+INP(PORT%)*256
120 IF C<CNT%   GOTO 90
130 'Print lapsed time and frequency
140 T=TIMER-T1
150 PRINT T,CNT%/T
160 END
```

## Appendix A: I/O Port Address Map

| I/O address range (Hex) | Uses |
|---|---|
| 000-1FF | used by base system board |
| 200 | not used |
| 201 | game control |
| 202-277 | not used |
| 278-27F | second printer port |
| 280-2F7 | not used |
| 2F8-2FF | COM2 |
| 300-377 | not used |
| 378-37F | printer port |
| 380-3AF | not used |
| 3B0-3BF | monochrome and printer |
| 3C0-3CF | not used |
| 3D0-3DF | color & graphics |
| 3E0-3EF | not used |
| 3F0-3F7 | 5 1/4 inch diskette drive |
| 3F8-3FF | COM1 |

# ภาคผนวก ข. การใช้งานไอซี LM 629

National Semiconductor

February 1995

# LM628/LM629 Precision Motion Controller

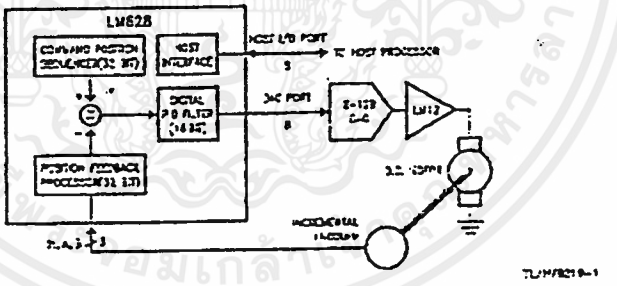
## General Description

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 N-MOS and packaged in a 28-pin dual in-line package or a 24-pin surface mount package (LM629 only). Both 5 MHz and 8 MHz maximum frequency versions are available with the suffixes -8 and -8, respectively, used to designate the versions. They incorporate an SDA core processor and cells designed by SDA.

## Features

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

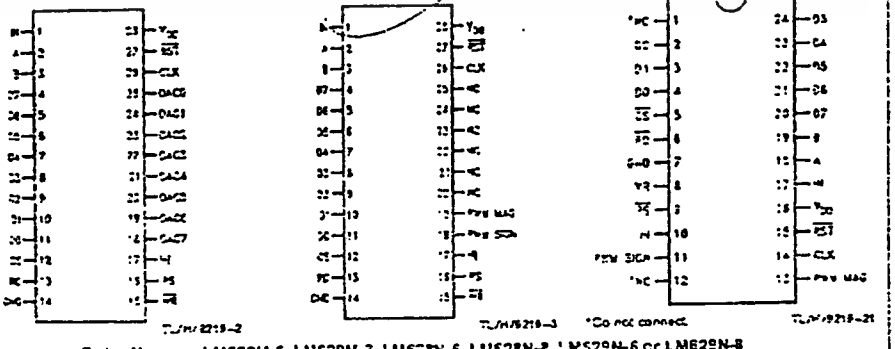





TL/H/8219–1

FIGURE 1. Typical System Block Diagram

## Connection Diagrams




TL/H/8219–2 TL/H/8219–3 TL/H/8219–21

Order Number LM628M-8, LM629M-8, LM628N-8, LM628N-8, LM629N-8 or LM629N-8
See NS Package Number M24B or N28B

## Absolute Maximum Ratings (Note 1)

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

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

## Operating Ratings

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

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

| Symbol | Parameter | Conditions | Tested Limits Min | Tested Limits Max | Units |
|---|---|---|---|---|---|
| $I_{DD}$ | Supply Current | Outputs Open | | 110 | mA |
| **INPUT VOLTAGES** | | | | | |
| $V_{IH}$ | Logic 1 Input Voltage | | 2.0 | | V |
| $V_{IL}$ | Logic 0 Input Voltage | | | 0.8 | V |
| $I_{IN}$ | Input Current | 0 < $V_{IN}$ < $V_{DD}$ | −10 | 10 | µA |
| **OUTPUT VOLTAGES** | | | | | |
| $V_{OH}$ | Logic 1 | $I_{OH}$ = −1.6 mA | 2.4 | | V |
| $V_{OL}$ | Logic 0 | $I_{OL}$ = 1.6 mA | | 0.4 | V |
| $I_{OUT}$ | TRI-STATE® Output Leakage Current | 0 < $V_{OUT}$ < $V_{DD}$ | −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 | Tø | 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 | | | | |
| LM629N-6, LM629N-6, LM629M-6 | T4 | 78 | | ns |
| LM629N-8, LM629N-8, LM629M-8 | T4 | 57 | | ns |
| Clock Period | | | | |
| LM629N-6, LM629N-6, LM629M-6 | T5 | 166 | | ns |
| LM629N-8, LM629N-8, LM629M-8 | T5 | 125 | | ns |
| Reset Pulse Width | T6 | $\frac{8}{f_{CLK}}$ | | µs |

## AC Electrical Characteristics (Continued)

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

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

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

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

Note 3: In order to read the busy bit, the status byte must first be read. The time required to read the busy bit far exceeds the time the user requires to test 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.

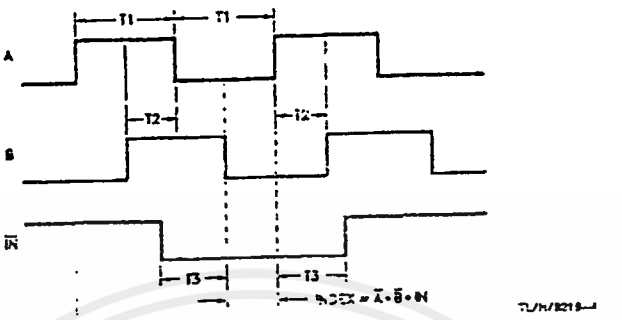



FIGURE 2. Quadrature Encoder Input Timing

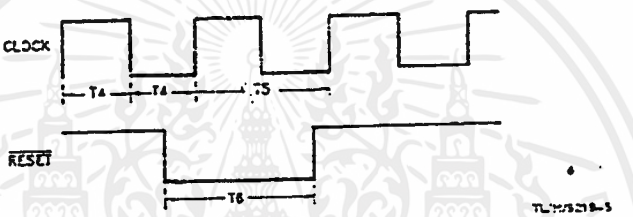



FIGURE 3. Clock and Reset Timing

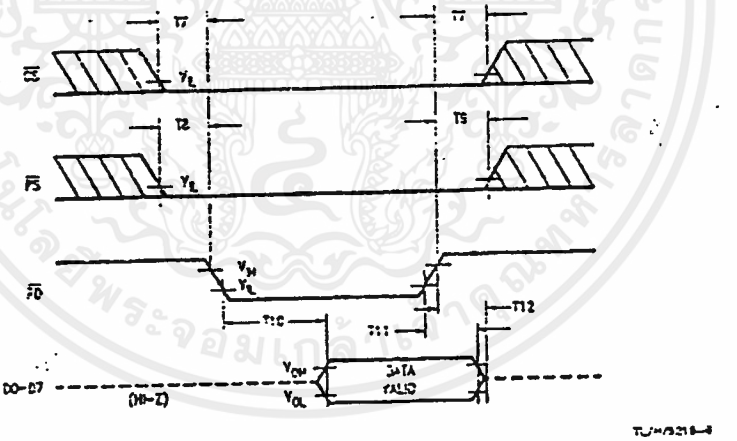



FIGURE 4. Status Byte Read Timing

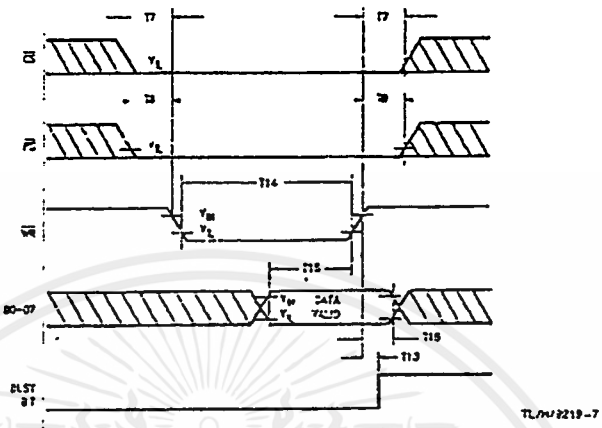


FIGURE 5. Command Byte Write Timing

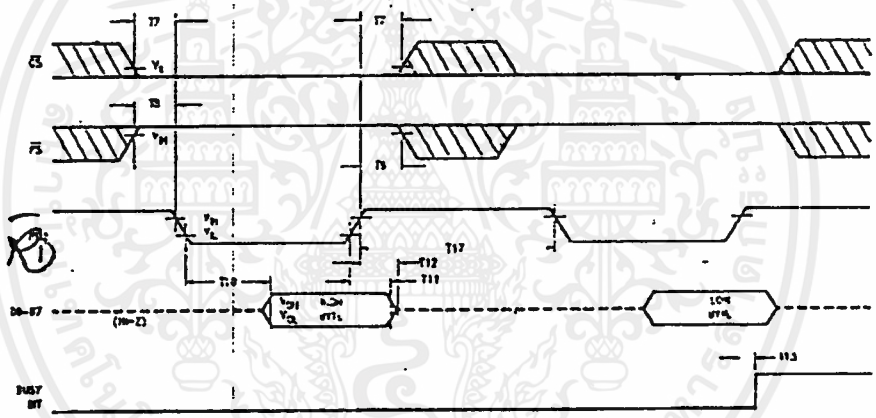


FIGURE 6. Data Word Read Timing

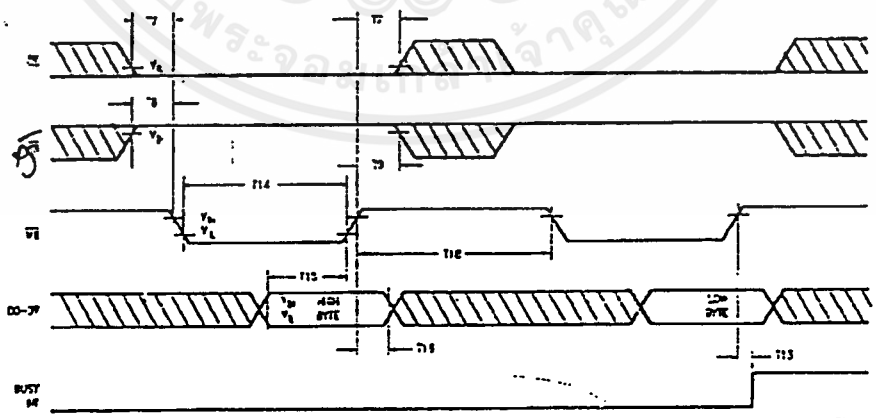


FIGURE 7. Data Word Write Timing

## Pinout Description

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

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

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

**Pins 4 to 11 (22 to 24 and 2 to 4), Host I/O Port (D0 to D7):** Bi-directional data port which connects to host computer/processor. Used for writing commands and data to the LM628, and for reading the status byte and data from the LM628, as controlled by $\overline{CS}$ (Pin 12), $\overline{PS}$ (Pin 16), $\overline{RD}$ (Pin 13), and $\overline{WR}$ (Pin 15).

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

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

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

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

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

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

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

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

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

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

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

1. Filter coefficient and trajectory parameters are zeroed.
2. Sets position error threshold to maximum value (7FFF hex), and effectively executes command UFEL.
3. The SBPA/SBPR interrupt is masked (disabled).
4. The five other interrupts are unmasked (enabled).
5. Initializes current position to zero as "home" position.
6. Sets derivative sampling interval to 2349/$f_{CLK}$ or 293 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 'C0'. 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 'C0' to '84' or 'C4' within 1.5 ms, perform another reset and repeat the above steps. To be certain that the reset was properly performed, execute a RSTI command. If the chip has reset properly, the status byte will change from hex '84' or 'C4' to hex '80' or 'C0'. If this does not occur, perform another reset and repeat the above steps.

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

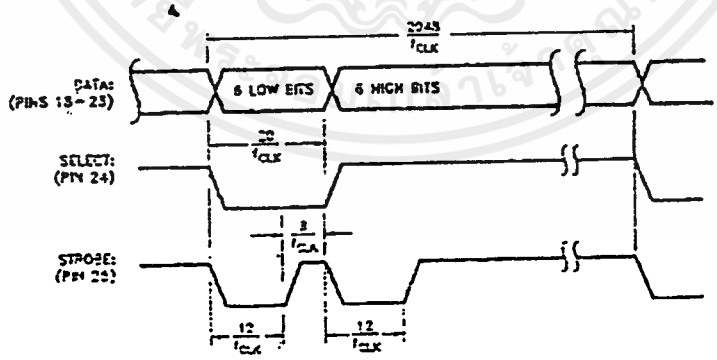



FIGURE 8. 12-Bit Multiplexed Output Timing

## Theory of Operation

### INTRODUCTION

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

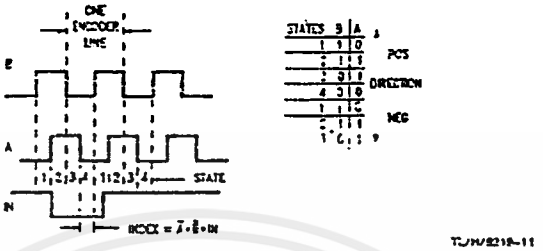
The optional index pulse output provided by some encoders assumes the logic-low state once per revolution. If the LM628 is so programmed by the user, it will record the absolute motor position in a dedicated register (the index register) at the time when all three encoder inputs are logic low.

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. Permanently grounding the index input will cause the LM628 to malfunction.

**TABLE I. System Specifications Summary**

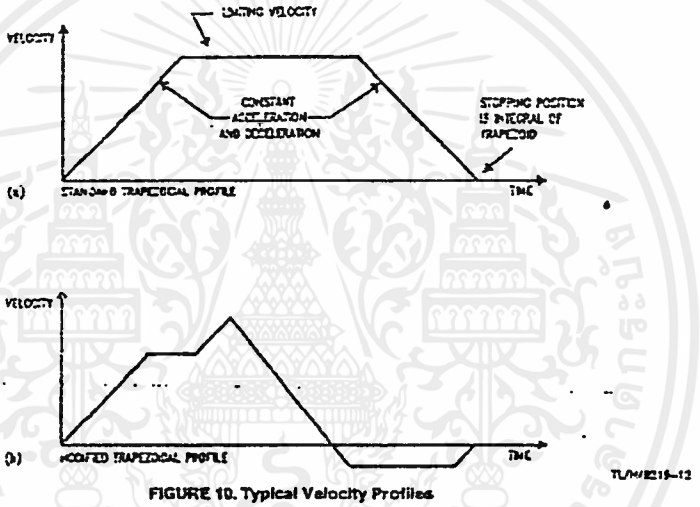
| | |
|---|---|
| Position Range | −1,073,741,824 to 1,073,741,823 counts |
| Velocity Range | 0 to $1,073,741,823/2^{15}$ counts/sample; ie. 0 to 16,383 counts/sample, with a resolution of $1/2^{15}$ counts/sample |
| Acceleration Range | 0 to $1,073,741,823/2^{16}$ counts/sample/sample; ie. 0 to 16,383 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<br>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).<br>Proportional and Integral: $2048/f_{CLK}$ |

## Theory of Operation (Continued)




TL/H/8219–11

FIGURE 9. Quadrature Encoder Signals




TL/H/8219–12

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

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

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

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 proportional 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 the 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 15) 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 15) 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 15) 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)

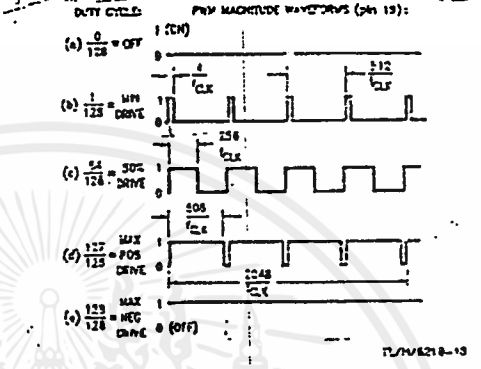
busy bit is logic low, the user may then sequentially transfer both bytes comprising a data word, but the busy bit must again be checked and found to be low before attempting to transfer the next byte pair (when transferring multiple words). Data transfers are accomplished via LM628-internal interrupts (which are not nested); the busy bit informs the host processor when the LM628 may not be interrupted for data transfer (or a command byte). If a command is written when the busy bit is high, the command will be ignored.

The busy bit goes high immediately after writing a command byte, or reading or writing a second byte of data (See *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 LM629 provides 8-bit, sign and magnitude PWM output signals for directly driving switch-mode motor-drive amplifiers. *Figure 11* shows the format of the PWM magnitude output signal.




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 | 3 |
| RDSTAT | Report | Read Status Byte | None | 1 | 1, 4 |
| RDSIGS | Report | Read Signals Register | 0C | 2 | 1 |
| RDIP | Report | Read Index Position | 09 | 4 | 1 |
| RDDP | Report | Read Desired Position | 08 | 4 | 1 |
| RDRP | Report | Read Real Position | 0A | 4 | 1 |
| RDDV | Report | Read Desired Velocity | 07 | 4 | 1 |
| RDRV | Report | Read Real Velocity | 0B | 2 | 1 |
| RDSUM | Report | Read Integration Sum | 0D | 2 | 1 |

Note 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 locally supported by hardware.

## User Command Set

**GENERAL**

The following paragraphs describe the user command set of the LM629. 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 LM629 user commands are used primarily to initialize the system for use.

**RESET COMMAND: RESET the LM629**

| | |
|---|---|
| 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 SBFA/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 defines 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 used 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 arm (or 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 5I. 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

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

(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: | 00000000 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 32kB/8 MHz sample interval must be scaled for other clock frequencies.

## Trajectory Control Commands (Continued)

The trajectory control parameters which are written to the LM628 to control motion are: acceleration, velocity, and position. In addition, indications as to whether these three parameters are to be considered as absolute or relative inputs, selection of velocity mode and direction, and manual stopping mode selection and execution are programmable via this command. After writing the command code, the first two data bytes that are written specify which parameter(s) is/are being changed. The first byte written is the more significant. Thus the two data bytes constitute a trajectory control word that informs the LM628 as to the nature and number of any following data bytes. See Table 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 |
| 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 compose 16-bit words. Each data item (parameter) requires two 16-bit words; the word and byte order is most-to-least significant. The order of sending the parameters to the LM628 corresponds to the descending order shown in the above description of the trajectory control word; i.e., beginning with acceleration, then velocity, and finally position.

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

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

STT COMMAND: STarT Motion Control

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

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

## Data Reporting Commands

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

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

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

**RDDP COMMAND: ReaD Desired Position**

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

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

**RDRP COMMAND: ReaD Real Position**

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

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

**RDDV COMMAND: ReaD Desired Velocity**

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

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

**RDRV COMMAND: ReaD Real Velocity**

| | |
|---|---|
| Command Code: | 08 Hex |
| Bytes Read: | Two |
| Data Range: | C000 to 3FFF Hex, See Text |
| Executable During Motion: | Yes |

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

**RDSUM COMMAND: ReaD Integration-Term SUMmation Value**

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

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

## Typical Applications

**Programming LM628 Host Handshaking (Interrupts)**

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

## 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 $\overline{PS}$ input. *Figure 12* also shows an 8-bit DAC and an LM12 Power Op Amp interfaced to the LM628.

### LM628 and High Performance Controller (HPC) Interface

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

### Interfacing a 12-Bit DAC

*Figure 14* illustrates use of a 12-bit DAC with the LM628. The 74LS378 hex gated-D flip-flop and an inverter demultiplex the 12-bit output. DAC offset must be adjusted to minimize DAC linearity and monotonicity errors. Two methods exist for making this adjustment. If the DAC1210 has been socketed, remove it and temporarily connect a 15 kΩ resistor between Pins 11 and 13 of the DAC socket (Pins 2 and 6 of the LF355) 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. 1, p. 2-280.

### Typical PWM Motor Drive Interfaces

*Figure 16* shows an LM18293 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)

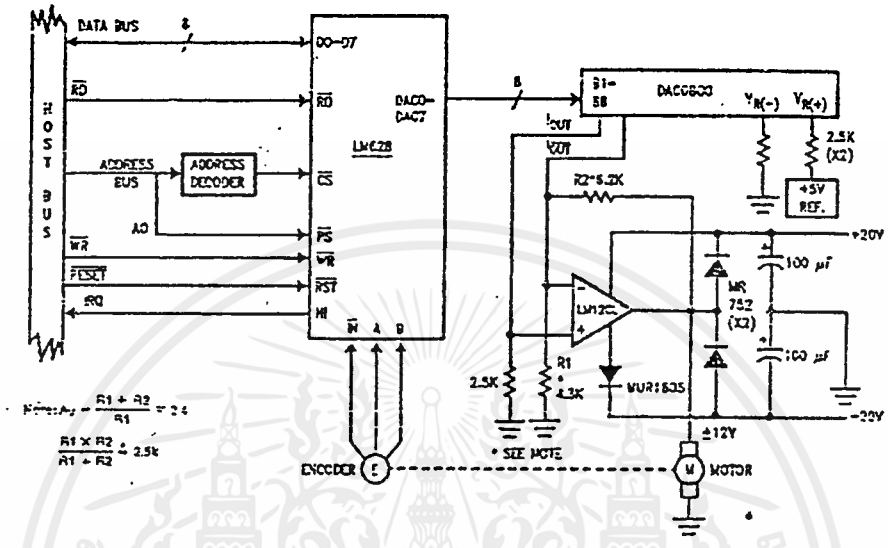



FIGURE 12. Host Interface and Minimum System Configuration

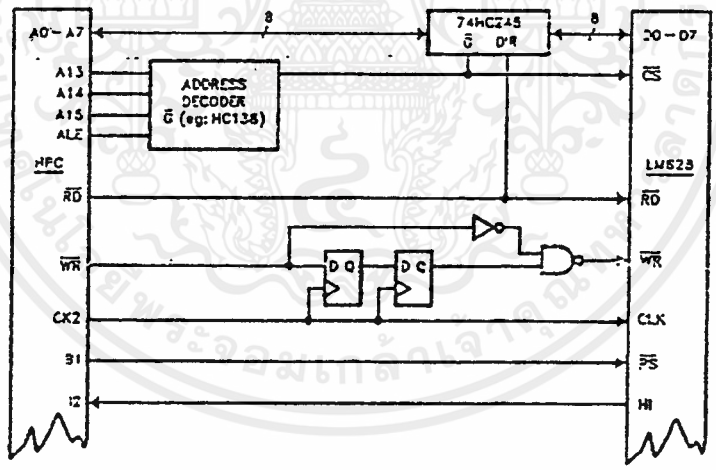
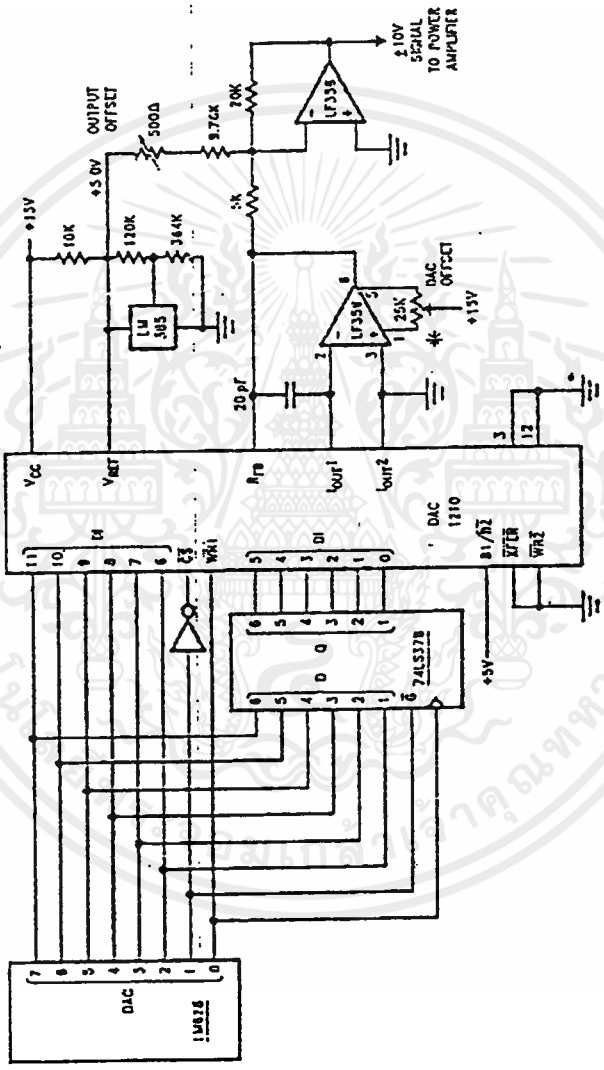



FIGURE 13. LM628 and HPC Interface

## Typical Applications (Continued)

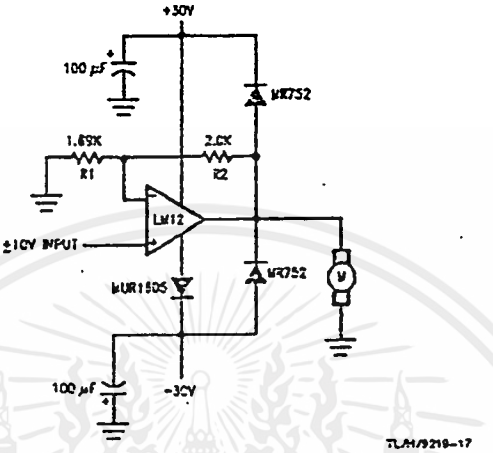



TL/H/5210-16

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

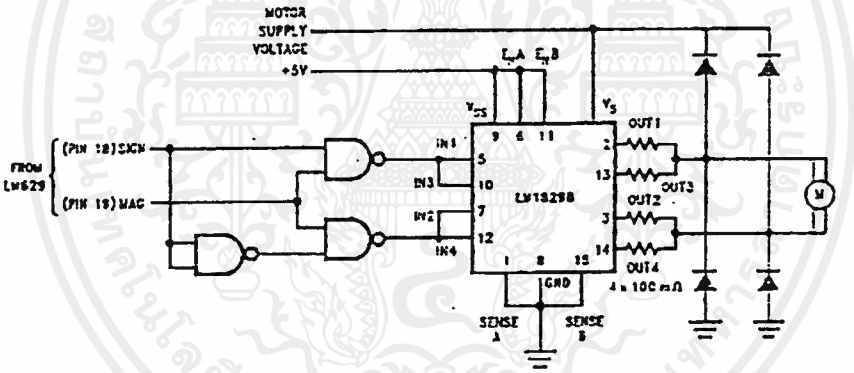
FIGURE 14. Interfacing a 12-Bit DAC and LM826

## Typical Applications (Continued)




TL/H/9218–17

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




TL/H/9218–18

FIGURE 16. PWM Drive for Brush/Commutator Motors

## Typical Applications (Continued)

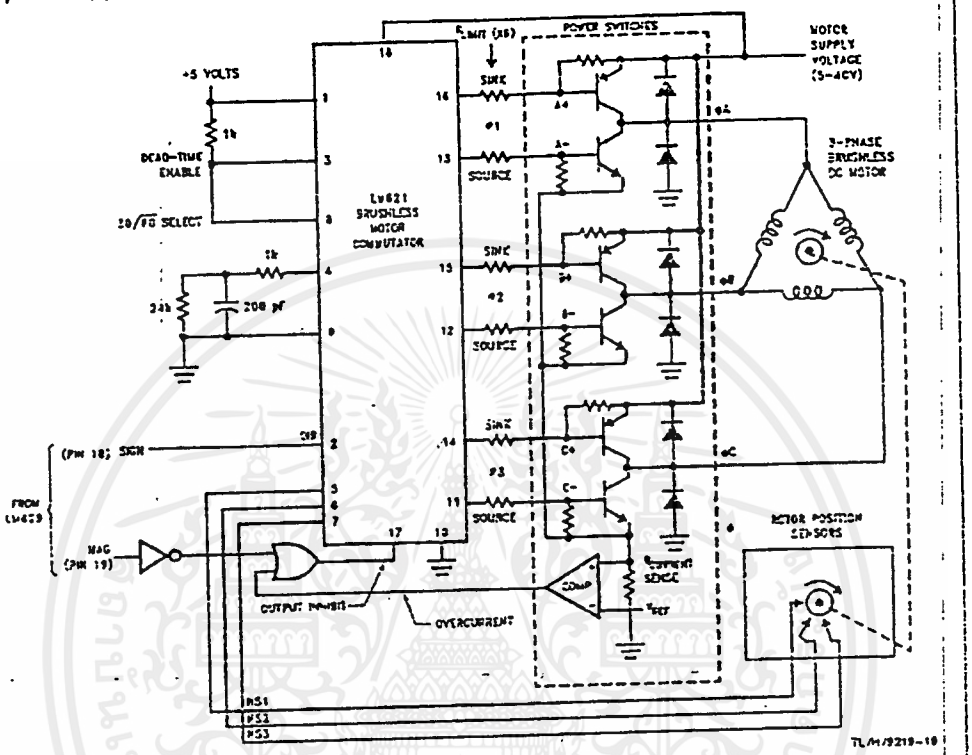



FIGURE 17. PWM Drive for Brushless Motors

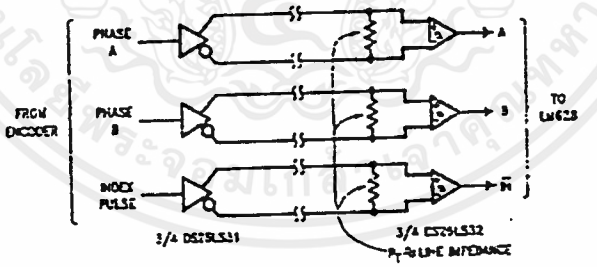
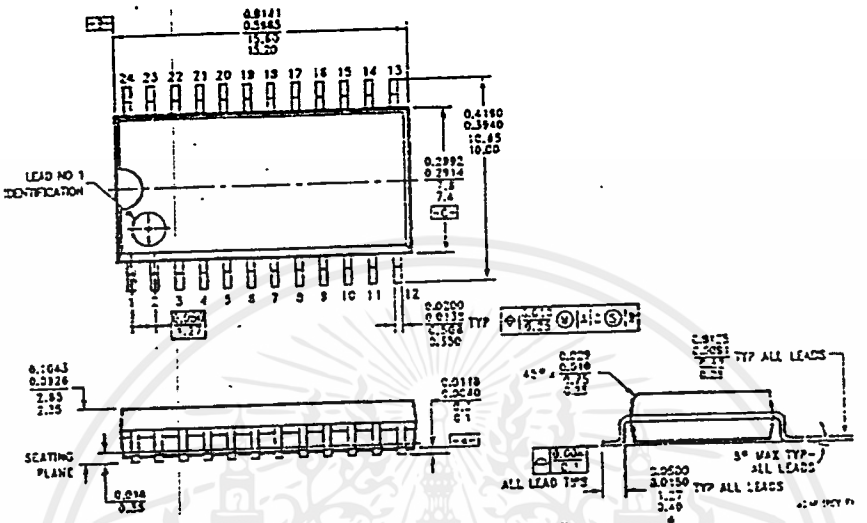


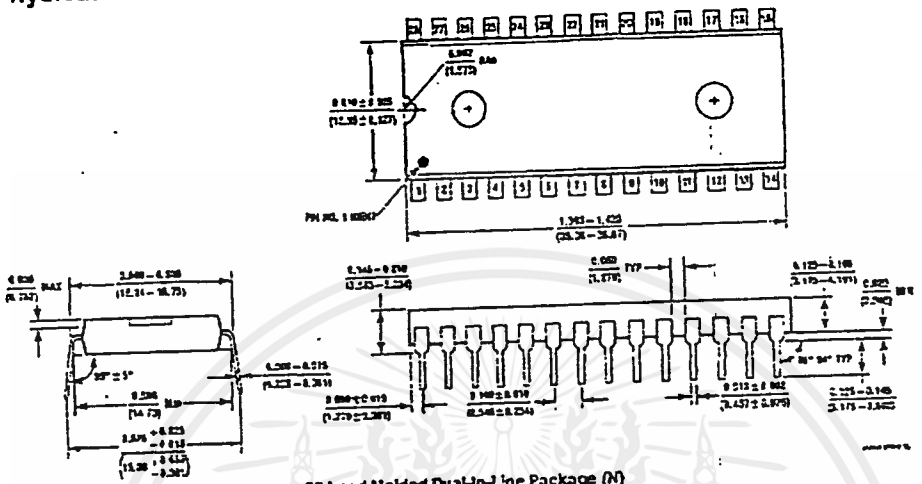

FIGURE 18. Typical Balanced-Line Encoder Input Circuit

## Physical Dimensions inches (millimeters)




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

## Physical Dimensions inches (millimeters) (Continued)



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

LIFE SUPPORT POLICY

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

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.

2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor
Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

National Semiconductor
Europe
Fax: (+49) 0-180-530 85 86
Email: [illegible]
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32
Français Tel: (+49) 0-180-532 93 58
Italiano Tel: (+49) 0-180-534 16 80

National Semiconductor
Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductor
Japan Ltd.
Tel: 81-043-299-2309
Fax: 81-043-299-2408

National Semiconductor

April 1992

# LM18298 Dual Full-Bridge Driver

## General Description

The LM18298 is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to gate the input control signals.

The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of a current sensing resistor. An additional supply input is provided to accommodate conventional logic supply voltages.

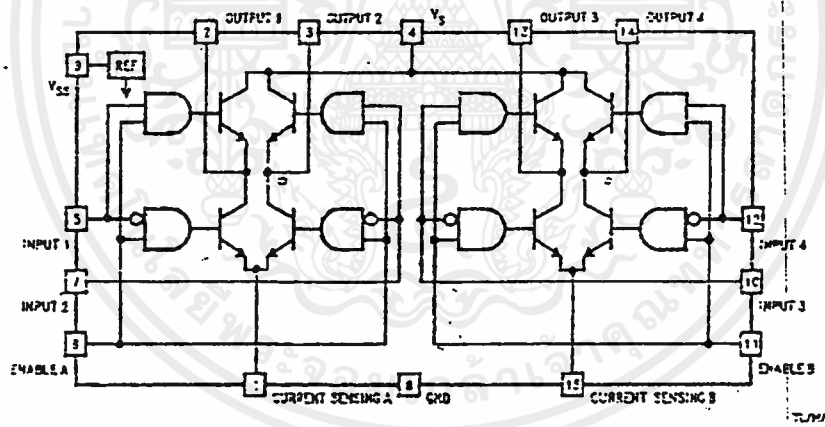
## Features

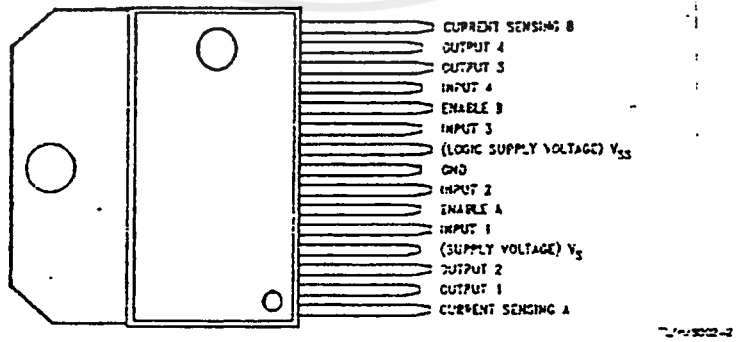
- Power supply voltage up to 46V
- 2A output per channel
- Low saturation voltage
- Thermal shutdown protection
- Logical "0" input voltage up to 1.5V (High noise immunity)
- Pin for pin replacement for L298N

## Applications

- DC and stepper motor drivers
- Relay and solenoid drivers

## Block & Connection Diagrams




TO 220-15
Order Number LM18298T
NS Package Number TA15A

## Absolute Maximum Ratings (Note 1)

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

| | |
|---|---|
| Main Supply (Pin 4) | 50V |
| Logic Supply (Pin 9) | 7V |
| Logic Inputs (Pins 5, 6, 7, 10, 11, 12) | −0.3 to 7V |
| Peak Output Current (Per Channel) | |
| Non-Repetitive (t = 100 µs) | 3A |
| Repetitive (80% duty cycle, $t_{ON}$ = 10 ms) | 2.5A |
| DC Operation | 2A |
| Sense Voltage (Pins 1, 15) | −1 to +2.3V |
| Power Dissipation (Note 2) | 25W |
| ESD Susceptibility (Note 3) | 1 kV |
| Lead Temperature (Soldering, 10 seconds) | 250°C |
| Storage Temperature Range | −65°C to +150°C |

## Operating Ratings

| | |
|---|---|
| Junction Temperature Range ($T_J$) | −40°C to +150°C |
| Main Supply (Pin 4) | 46V |

## Electrical Characteristics

$V_S$ = 42V, $V_{SS}$ = 5V, $I_O$ = 0A, $T_J$ = 25°C, L = 0V, H = 5V, unless otherwise specified

| Symbol | Parameter | Conditions | Typical (Note 4) | Limit (Note 5) | Units (Limits) |
|---|---|---|---|---|---|
| $V_S$ | Main Supply Voltage (Pin 4) | | | $V_{SS}$ + 2.5 | V (min) |
| | | | | 46 | V (max) |
| $V_{SS}$ | Logic Supply Voltage (Pin 9) | | | 4.5 | V (min) |
| | | | | 7 | V (max) |
| $I_S$ | Main Supply Quiescent Current (Pin 4) | Enable = H, Input = L | 8 | 22 | mA (max) |
| | | Enable = H, Input = H | 32 | 70 | |
| | | Enable = L, Input = X | | 4 | |
| $I_{SS}$ | Logic Supply Quiescent Current (Pin 9) | Enable = H, Input = L | 22 | 36 | mA (max) |
| | | Enable = H, Input = H | 8 | 12 | |
| | | Enable = L, Input = X | | 6 | |
| $V_{IL}$ | Low Level Input Voltage (Pins 5, 7, 10, 12) | | | −0.3 | V (min) |
| | | | | 1.5 | V (max) |
| $V_{IH}$ | High Level Input Voltage (Pins 5, 7, 10, 12) | | | 2.3 | V (min) |
| | | | | $V_{SS}$ | V (max) |
| $I_{IL}$ | Low Level Input Current (Pins 5, 7, 10, 12) | Input = L | | −10 | µA (max) |
| $I_{IH}$ | High Level Input Current (Pins 5, 7, 10, 12) | Input = H | 30 | 100 | µA (max) |
| $V_{ENL}$ | Low Level Enable Voltage (Pins 6, 11) | | | −0.3 | V (min) |
| | | | | 1.5 | V (max) |
| $V_{ENH}$ | High Level Enable Voltage (Pins 6, 11) | | | 2.3 | V (min) |
| | | | | $V_{SS}$ | V (max) |
| $I_{ENL}$ | Low Level Enable Input Current (Pins 6, 11) | Enable = L | | −10 | µA (max) |
| $I_{ENH}$ | High Level Enable Input Current (Pins 6, 11) | Enable = H | 30 | 100 | µA (max) |

## Electrical Characteristics (Continued)

$V_S$ = 42V, $V_{SS}$ = 5V, $I_O$ = 0A, $T_J$ = 25°C, unless otherwise specified

| Symbol | Parameter | Conditions | Typical (Note 4) | Limit (Note 5) | Units (Limits) |
|---|---|---|---|---|---|
| $V_{CE\ sat(H)}$ | Source Saturation Voltage (Pins 2, 3, 13, 14) | $I_O$ = 1A | 1.35 | 1.7 | V (max) |
| | | $I_O$ = 2A | 2.0 | 2.7 | |
| $V_{CE\ sat(L)}$ | Sink Saturation Voltage (Pins 2, 3, 13, 14) | $I_O$ = 1A | 1.2 | 1.6 | V (max) |
| | | $I_O$ = 2A | 1.7 | 2.3 | |
| $V_{CE\ sat}$ | Total Drop $V_{CE\ sat(H)}$ + $V_{CE\ sat(L)}$ | $I_O$ = 1A | | 3.2 | V (max) |
| | | $I_O$ = 2A | | 4.9 | |
| $V_{sense}$ | Sensing Voltage (Pins 1, 15) | t < 50 µs | | −1 | V (min) |
| | | Continuous | | −0.5 | |
| | | Continuous | | 2 | V (max) |
| $T_1$ | Source Current Turn-Off Delay | 0.5 Input to 0.9 $I_O$ (*Figure 2*) | 0.5 | | µs |
| $T_2$ | Source Current Fall Time | 0.9 $I_O$ to 0.1 $I_O$ (*Figure 2*) | 0.15 | | µs |
| $T_3$ | Source Current Turn-On Delay | 0.5 Input to 0.1 $I_O$ (*Figure 2*) | 1.3 | | µs |
| $T_4$ | Source Current Rise Time | 0.1 $I_O$ to 0.9 $I_O$ (*Figure 2*) | 0.85 | | µs |
| $T_5$ | Sink Current Turn-Off Delay | 0.5 Input to 0.9 $I_O$ (*Figure 3*) | 0.25 | | µs |
| $T_6$ | Sink Current Fall Time | 0.9 $I_O$ to 0.1 $I_O$ (*Figure 3*) | 0.1 | | µs |
| $T_7$ | Sink Current Turn-On Delay | 0.5 Input to 0.1 $I_O$ (*Figure 3*) | 1.3 | | µs |
| $T_8$ | Sink Current Rise Time | 0.1 $I_O$ to 0.9 $I_O$ (*Figure 3*) | 0.1 | | µs |
| $f_C$ | Commutation Frequency | $I_O$ = 2A | 25 | | kHz |

**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 its specified Operating Ratings.

**Note 2:** The maximum power dissipation must be derated at elevated temperatures and is a function of $T_{J\ max}$, $\theta_{JC}$, and $T_C$. The maximum allowable power dissipation at any temperature is $P_{D\ max} = (T_{J\ max} - T_C)/\theta_{JC}$ or the number given in the Absolute Maximum Ratings, whichever is lower. The typical junction-to-case thermal resistance ($\theta_{JC}$) of the LM18298 is 3°C/W.

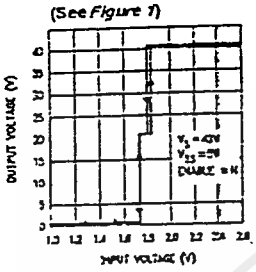
**Note 3:** Human body model, 100 pF discharged through a 1.5 kΩ resistor.

**Note 4:** Typicals are at 25°C and represent the most likely parametric norm.

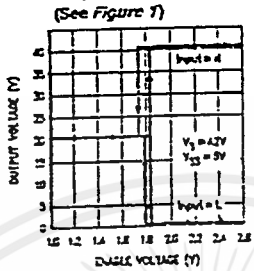
**Note 5:** Limits are guaranteed and 100% tested.

## Typical Performance Characteristics

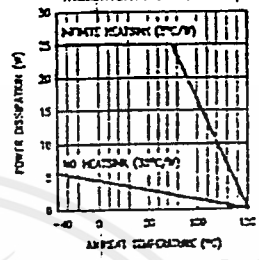
Output Voltage
(See *Figure 1*)



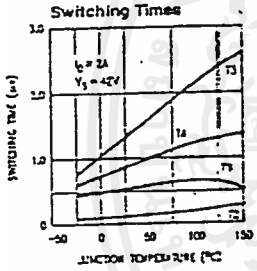

Output Voltage
(See *Figure 1*)



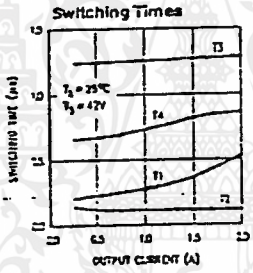

Maximum Power Dissipation



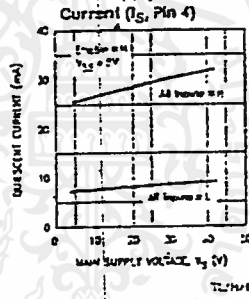

Source Current
Switching Times



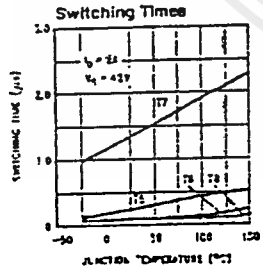

Source Current
Switching Times



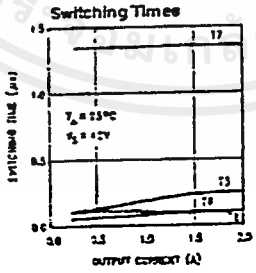

Main Supply Quiescent
Current ($I_S$, Pin 4)



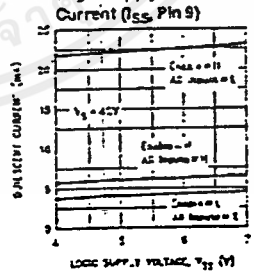

Sink Current
Switching Times




Sink Current
Switching Times




Logic Supply Quiescent
Current ($I_{SS}$, Pin 9)



## Test Circuits

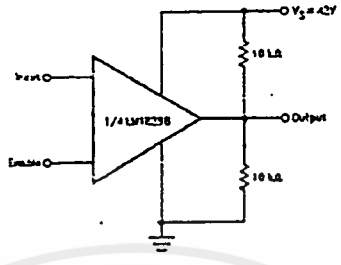


FIGURE 1. Input/Enable Threshold Test Circuit

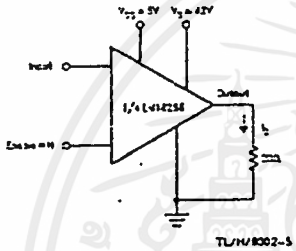


FIGURE 2(a). Source Current Switching Time Test Circuit

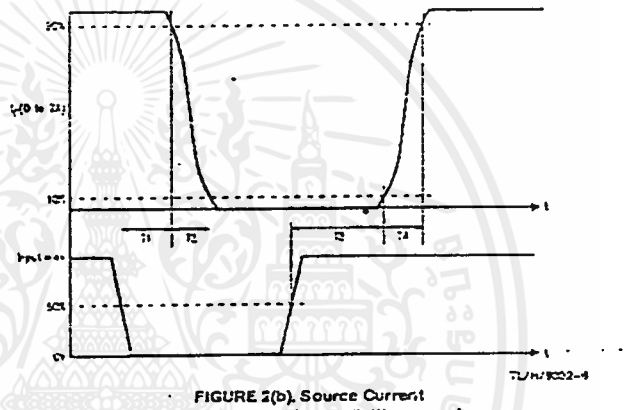


FIGURE 2(b). Source Current Switching Time Definitions

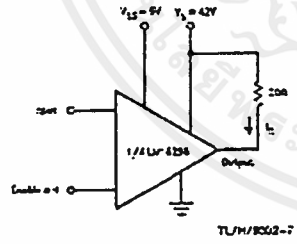


FIGURE 3(a). Sink Current Switching Time Test Circuit

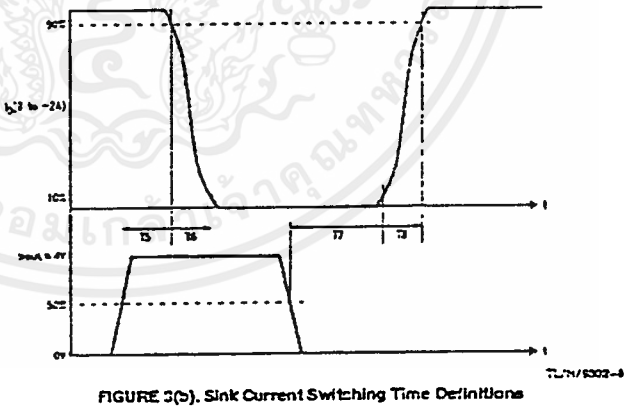
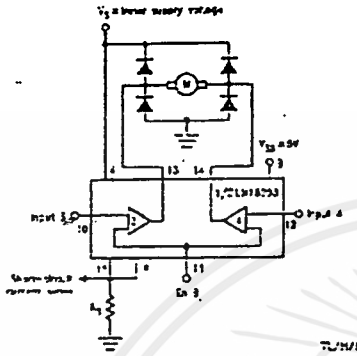


FIGURE 3(b). Sink Current Switching Time Definitions

## Applications Information

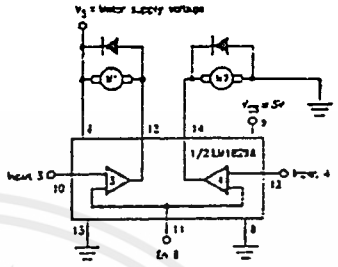


TL/H/5002–9

| Enable B | Inputs | Motor Direction |
|---|---|---|
| H | Input 3 = H, Input 4 = L | Clockwise |
| | Input 3 = L, Input 4 = H | Counterclockwise |
| | Input 3 = Input 4 | Dynamic Braking |
| L | Input 3 = X, Input 4 = Input 3 | Coast to a Stop |

L = Low, H = High X = don't care

FIGURE 4. Bidirectional DC Motor Control

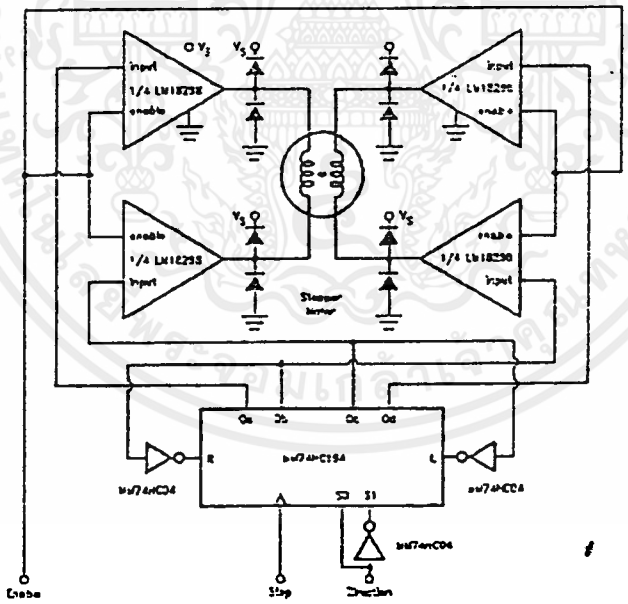


TL/H/5002–10

| Enable B | Input 3 | Motor 1 | Input 4 | Motor 2 |
|---|---|---|---|---|
| H | H | Dynamic Braking | H | Run |
| H | L | Run | L | Dynamic Braking |
| L | X | Coast to a Stop | X | Coast to a Stop |

L = Low H = High X = Don't Care

FIGURE 5. 2-Motor Controller
(Using both High- and Low-Side Driver Modes)



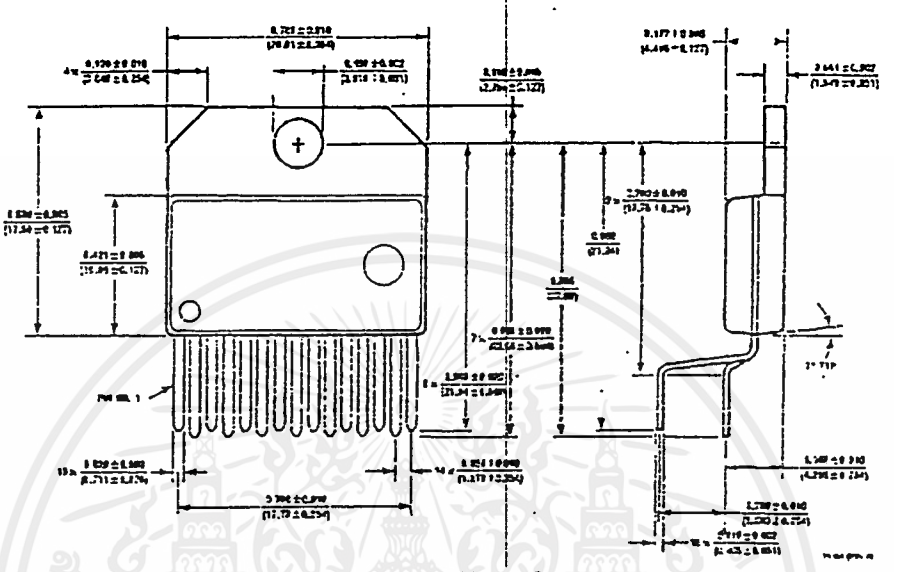
TL/H/5002–11

FIGURE 6. Two-Phase Bipolar Stepper Motor Control Circuit

**CLAMP DIODES**

When driving inductive loads, diodes are necessary to clamp spikes at the LM18293 outputs. Clamp diodes must have a recovery time of 200 ns or better and a forward drop of 1.2V or less at the rated load current. Typical devices are the MB346 (Microsemi Corp., Santa Ana, CA), and the V331X (Varo Semiconductor Inc., Garland, TX).

## Physical Dimensions inches (millimeters)



Order Number LM18298T
See NS Package Number TA15A

### LIFE SUPPORT POLICY

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

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.

2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

National Semiconductor Europe
Fax: (+49) 0-180-530 85 86
Email: onwge@tevm2.nsc.com
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32
Français Tel: (+49) 0-180-532 93 58
Italiano Tel: (+49) 0-180-534 16 80

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductor Japan Ltd.
Tel: 81-043-299-2309
Fax: 81-043-299-2408

# ภาคผนวก ค.โปรแกรมที่ใช้ในการคำนวนค่าเกนป้อนกลับ

%โปรแกรมนี้ ใช้สำหรับคำนวนค่า K ที่เปลี่ยนแปรไปตามค่า ความยาวเชือกที่เปลี่ยนไป
%โดยจะคำนวนที่ค่าความยาวเชือกตั้งแต่ 1.00 เมตร ลดลงไปจนถึง 0.01 เมตร
%โดยโปรแกรมนี้จะให้ค่าความเร็วของการดึงเชือกเป็น 0 เมตร/วินาที

```
datafile = fopen('calk0a.dat','wt+');
K=[0 0 0 0 0 ]';I=1;
L=1;
for n = 1:100,

  A=[0 1 0 0 0;-9.81/L 0 0 0 -1/L;0 0 0 1 0;0 0 0 0 1;0 0 0 -49 -12.6];

  B=[0 0 0 0 49*7/8]';
  C=[0 0 1 0 0];

  [G,H] = c2d(A,B,0.1);
  Q=diag([1000 1 1000 1 1]); R=(200);
  [K,S,E]=dlqI(G,H,Q,R);
K1=K(1);
K2=K(2);
K3=K(3);
K4=K(4);
K5=K(5);
bL=100*L;
L=L-0.01;
fprintf(datafile,'case %3.0f :Kf.angle=%2.4f ; Kf.difAngle=%2.4f ; Kf.pos=%2.4f ; Kf.d
ifPos=%2.4f ;Kf.difDifPos= %2.4f;break; \n',bL,K1,K2,K3,K4,K5);
end
fclose(datafile);
%load c:\matlab\toolbox\control\calk0a.dat;
```

```
%โปรแกรมนี้ ใช้สำหรับคำนวนค่า K ที่เปลี่ยนแปรไปตามค่า ความยาวเชือกที่เปลี่ยนไป
%โดยจะคำนวนที่ค่าความยาวเชือกตั้งแต่ 1.00 เมตร ลดลงไปจนถึง 0.01 เมตร
%โดยโปรแกรมนี้จะให้ค่าความเร็วของการดึงเชือกเป็น -0.5 เมตร/วินาที
% ซึ่งหมายถึงการดึงขึ้น

datafile = fopen('calk_1.dat','wt+');
K=[0 0 0 0 0 ]';I=1;
difL=-0.5;
L=1;
for n = 1:100,

   A=[0 1 0 0 0;-9.81/L -2*difL/L 0 0 -1/L;0 0 0 1 0;0 0 0 0 1;0 0 0 -49 -12.6];

   B=[0 0 0 0 49*7/8]';
   C=[0 0 1 0 0];

   [G,H] = c2d(A,B,0.1);
   Q=diag([1000 1 1000 1 1]);R=(200);
   [K,S,E]=dlqr(G,H,Q,R);
K1=K(1);
K2=K(2);
K3=K(3);
K4=K(4);
K5=K(5);
bL=100*L;
L=L-0.01;

fprintf(datafile,'case %3.0f :Kf.angle=%2.4f ; Kf.difAngle=%2.4f ; Kf.pos=%2.4f ; Kf.d
ifPos=%2.4f ;Kf.difDifPos= %2.4f;break; \n',bL,K1,K2,K3,K4,K5);
end
fclose(datafile);
%load c:\matlab\toolbox\control\calk_1.dat;
```

```
%โปรแกรมนี้ ใช้สำหรับคำนวนค่า K ที่เปลี่ยนแปรไปตามค่า ความยาวเชือกที่เปลี่ยนไป
%โดยจะคำนวนที่ค่าความยาวเชือกตั้งแต่ 1.00 เมตร ลดลงไปจนถึง 0.01 เมตร
%โดยโปรแกรมนี้จะให้ค่าความเร็วของการดึงเชือกเป็น 0.5 เมตร/วินาที
%ซึ่งหมายถึงการดึงลง

datafile = fopen('calk1.dat','wt+');
K=[0 0 0 0 0 ]';i=1;
difL=0.5;
L=1;
for n = 1:100,

   A=[0 1 0 0 0;-9.81/L -2*difL/L 0 0 -1/L;0 0 0 1 0;0 0 0 0 1;0 0 0 -49 -12.6];

   B=[0 0 0 0 49*7/8]';
   C=[0 0 1 0 0];

   [G,H] = c2d(A,B,0.1);
   Q=diag([1000 1 1000 1 1]); R=(200);
   [K,S,E]=dlqr(G,H,Q,R);
K1=K(1);
K2=K(2);
K3=K(3);
K4=K(4);
K5=K(5);
bL=100*L;
L=L-0.01;

fprintf(datafile,'case %3.0f :Kf.angle=%2.4f ; Kf.difAngle=%2.4f ; Kf.pos=%2.4f ; Kf.d
ifPos=%2.4f ;Kf.difDifPos= %2.4f;break; \n',bL,K1,K2,K3,K4,K5);
end
fclose(datafile);
%load c:\matlab\toolbox\control\calk1.dat;
```

# ภาคผนวก ง. Source Code ของโปรแกรมที่ใช้ในการควบคุม

```
/* Program Test LQR&LQG Funtions
        Begin       24 MAR 1996
        Last Update 7  MAR 1997
*/

/* Standard Include Files */

#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

/* Ours Include Files */

#include <adc5.h>
#include <input629.h>
#include <lm629v2.h>
#include <lm629vc.h>
#include <plotgp13.h>

#define PERIOD0 0.020                   //sampling time in microsecond
#define PERIOD 0.1

#define ADD_RELAY 0x304
#define UP 0x01
#define DOWN 0x00
#define STOP_UP 0x05
#define STOP_DOWN 0x04

#define LOW1 2500                       //value to load to counter2 8253
#define PI 3.14159265359
```

```
#define LOW2 5000
#define X  16
#define Y  5
#define MAXDATA 600
#define craneFile       "cdlqr5.dat"    //response data file

typedef struct{
        float angle;
        float difAngle;
        float pos;
        float difPos;
        float difDifPos;
        }feedBackGain;


void main(void)
{
                /*define variable*/
        word adc;
        int mode=PACER;
        int base=BASE;   //0x220 is base of PCL-714
        int ch=12;
        counterRegister counter;
        float offset;
        float volt2angle=(30/3);


        graphInfo info;
        axisScale scale;
        float maxDistance=1.0;
        float samplingTime,maxTime=60.0;
```

```
int k=0;
int z=0,y=0;
float CHK;
long time=0;
long startProcess=0,startLoop=0,end=0;

trjType input;


float releasePos,stopPos=0.5,sPos=0.5;

float velocity=0.0;

float loadPos;
double radian;

FILE *datafile;

float volt[MAXDATA],realAngle[MAXDATA],realPos[MAXDATA] ;
float realDifAngle[MAXDATA],realDifPos[MAXDATA],realDifDifPos[MAXDATA];
float pos[MAXDATA],angle[MAXDATA];
float controlLaw[MAXDATA],velo[MAXDATA],desireVelo[MAXDATA];
float difPos[MAXDATA],difDifPos[MAXDATA],difAngle[MAXDATA];

float ropPos[MAXDATA];
float chgRopPos=0;
//float kPos,kDifPos,kDifDifPos,kAngle,kDifAngle;
feedBackGain Kf;
float f1,f2,f3,f4,f5;
int zz[MAXDATA];
float yy[MAXDATA];
```

```
        int Lrop;
        div_t buff;

        float errorPos[MAXDATA],errorAngle[MAXDATA];

        float ePos[MAXDATA],eAngle[MAXDATA];

        float filAngle[MAXDATA];
        float a1[5],b1[5],a2[5],b2[5],a3[5],b3[5],a4[5],b4[5],a5[5],b5[5],a6[5],b6[5];

        float posRef=1.3;       //insert the distance here
        float angleRef=0.0;
        float beginRopPos=0.85;
        float desiredRopPos=0.75;

/* start program */

        clrscr();

/* Info of graph */
        info.xMax=maxTime;
        info.yMax=maxDistance;
        info.sampling=PERIOD;
        info.heading="Response of Load Position";
        info.xLabel="Time (sec)";
        info.yLabel="Distance (cm)";
        samplingTime=info.sampling;

/* Initial data */
        for(k=0;k<MAXDATA;k++)
        {
        volt[k]=0;
        realAngle[k]=0;
```

```
realPos[k]=0;
realDifAngle[k]=0;
realDifPos[k]=0;
realDifDifPos[k]=0;

angle[k]=0;
difAngle[k]=0;
pos[k]=0;
difPos[k]=0;
difDifPos[k]=0;
ropPos[k]=0;

controlLaw[k]=0;
velo[k]=0;
desireVelo[k]=0;
zz[k]=0;

errorPos[k]=0;
errorAngle[k]=0;

ePos[k]=0;        /* realPos-estimatePos */
eAngle[k]=0;      /* realAngle-estimateAngle */

filAngle[k]=0;

}



/* Initial FeedBack Gain */
```

```
        k=0;

        counter=COUNTER_REGISTER(PERIOD*1000000);
        gotoxy(10,1);
        printf("TEST ALL PROGRAM\n");
        printf("This is Pacer Mode.\n");      //use pace mode to trig ADC

/* Initial LM629 */
        INIT_LM629();                         //initial lm629
        WRITE_FILTER();
        INIT_LM6292();                        //initial lm629
        WRITE_FILTER2();
        ACCELERATION_OUT(8);                  //initial acceleration

        scale = INITIAL_GRAPH(info);          //open graphic mode and initial to plot
graph


        SET_COUNTER(base,counter);            //load sampling time to counter registe
r and start counting


        startProcess=clock();                 //read start Process clock
        startLoop=startProcess;         //read start Loop clock
        do
        {

            /* Read Angle from ADC */
        TRIG(mode,base,ch,counter,k);   //select channel and trig ADC to read data
        adc=READ_ADC(base);             //wait for reading value from ADC when reach sa
 mpling time
```

```
        k++;

        volt[k]=(((adc.high*256)+adc.low)-8192);
        volt[k]=(volt[k]*10)/16384;          //converse data to voltage value
        if(k==1)
                        offset=volt[1];
        volt[k]=volt[k]-offset;                    //automatic adjust offset
        realAngle[k]=((volt[k]*volt2angle)*2*PI)/360;    //compute angle from voltage
to "radian"

                /* Filter Angle before Compute */

        a1[1]=1.0000; a1[2]=0.0309; a1[3]=-0.9691;
        b1[1]=0.9845; b1[2]=0.0000; b1[3]=-0.9845;
        a6[1]=1.0000; a6[2]=-0.9843;a6[3]=0.0157;
        b6[1]=0.4921; b6[2]=0.0000; b6[3]=-0.4921;

        if (k==1)
        {
                filAngle[k]=b1[1]*realAngle[k];
        }
        if (k==2)
        {
                filAngle[k]=b1[1]*realAngle[k]+b1[2]*realAngle[k-1]-a1[2]*filAngle[k-
1];
        }
        if ((k!=1)&&(k!=2))
        {
                filAngle[k]=b1[1]*realAngle[k]+b1[2]*realAngle[k-1]+b1[3]*realAngle[k
-2]-a1[2]*filAngle[k-1]-a1[3]*filAngle[k-2];
        }

                /* Read Position from lm629 */
```

```
realPos[k]=READ_REAL_POS();     //read real position from lm629
velo[k]=READ_REAL_VELO();       //read real velocity from lm629
desireVelo[k]=READ_DESIRED_VELO();

chgRopPos=READ_REAL_POS2();
ropPos[k]=beginRopPos-chgRopPos;


                /* module Loop for liflod*/
/* z and y is made for check the status */

if ((z==0)&&(y==0))
    {                       // z=1 relay was up
        outp(ADD_RELAY,UP);
        z++;
    }
     else if ((z==1)&&(ropPos[k]<=desiredRopPos))  //cal the stop Pos
     {
//   stopPos=(beginRopPos*realPos[k])/(beginRopPos-ropPos[k]);
        stopPos=realPos[k]+(posRef-realPos[k])/2;
        sPos=stopPos-0.2;
        z++ ;
     }
     else if ((z==2)&&(y==0)&&(realPos[k]>=sPos))
        {


                                            // z=3 relay was down
        outp(ADD_RELAY,STOP_UP);          // y=0 with no stopping time
        outp(ADD_RELAY,STOP_DOWN);
            if (realPos[k]>=stopPos)
            {
            outp(ADD_RELAY,DOWN);
            z++;
```

```
            }
        }
    else if ((z==2)&&(y==0)&&(ropPos[k]<=0.15))
    {                                   // y=1 there is some stoppingtime
    outp(ADD_RELAY,STOP_UP);
    outp(ADD_RELAY,STOP_DOWN);
    releasePos=realPos[k]+2*(stopPos-realPos[k]);
    y++;
        }
    else if ((z==2)&&(y==1)&&(realPos[k]>=releasePos))
    {
    outp(ADD_RELAY,DOWN);
    z++;                                // z=3  relay down
    y++;
    }
    else if ((z==3)&&(ropPos[k]>=desiredRopPos))
        {
        outp(ADD_RELAY,STOP_DOWN);          //  z=4 end //
        outp(ADD_RELAY,STOP_UP);
        z++;
        }
                    /*feedback checking module*/

         CHK=inp(ADD_RELAY) ;
        if ((z==1)&&(CHK!=UP))
        {
        outp(ADD_RELAY,UP);
        }
        else if ((z==3)&&(CHK!=DOWN))
        {
        outp(ADD_RELAY,DOWN);
        }
```

```
        /* Calculate DifAngle DifPos DifDifPos */


if (k==1)
{
        realDifPos[k]=0;
        realDifDifPos[k]=0;
        realDifAngle[k]=0;
}
else
{
        realDifPos[k]=(realPos[k]-realPos[k-1])/info.sampling;
        realDifAngle[k]=(filAngle[k]-filAngle[k-1])/info.sampling;
}
if (k==2)
{
        realDifDifPos[k]=0;
}
else
{
        realDifDifPos[k]=(realDifPos[k]-realDifPos[k-1])/info.sampling;
}


errorPos[k]=posRef-realPos[k];
errorAngle[k]=angleRef+filAngle[k];
Lrop=ropPos[k]*100;
buff = div(Lrop,2);
Lrop = buff.quot*2;
        /* LQR */

// a new method //
```

```
if (z==1)
{
        switch(Lrop){
// when lift up
/*case 100 :Kf.angle=-3.5084 ; Kf.difAngle=-0.5183 ; Kf.pos=1.7739 ; Kf.difPos=0.7689
;Kf.difDifPos= 0.0927;break;
case  99 :Kf.angle=-3.4944 ; Kf.difAngle=-0.5229 ; Kf.pos=1.7717 ; Kf.difPos=0.7665 ;K
f.difDifPos= 0.0931;break;
case  98 :Kf.angle=-3.4801 ; Kf.difAngle=-0.5275 ; Kf.pos=1.7694 ; Kf.difPos=0.7640 ;K
f.difDifPos= 0.0936;break;
case  97 :Kf.angle=-3.4655 ; Kf.difAngle=-0.5321 ; Kf.pos=1.7670 ; Kf.difPos=0.7615 ;K
f.difDifPos= 0.0941;break;
case  96 :Kf.angle=-3.4506 ; Kf.difAngle=-0.5367 ; Kf.pos=1.7646 ; Kf.difPos=0.7590 ;K
f.difDifPos= 0.0946;break;
case  95 :Kf.angle=-3.4354 ; Kf.difAngle=-0.5413 ; Kf.pos=1.7622 ; Kf.difPos=0.7564 ;K
f.difDifPos= 0.0951;break;
case  94 :Kf.angle=-3.4199 ; Kf.difAngle=-0.5458 ; Kf.pos=1.7597 ; Kf.difPos=0.7537 ;K
f.difDifPos= 0.0956;break;
case  93 :Kf.angle=-3.4041 ; Kf.difAngle=-0.5504 ; Kf.pos=1.7572 ; Kf.difPos=0.7510 ;K
f.difDifPos= 0.0961;break;
case  92 :Kf.angle=-3.3880 ; Kf.difAngle=-0.5549 ; Kf.pos=1.7547 ; Kf.difPos=0.7483 ;K
f.difDifPos= 0.0966;break;
case  91 :Kf.angle=-3.3715 ; Kf.difAngle=-0.5593 ; Kf.pos=1.7521 ; Kf.difPos=0.7455 ;K
f.difDifPos= 0.0972;break;
case  90 :Kf.angle=-3.3547 ; Kf.difAngle=-0.5638 ; Kf.pos=1.7494 ; Kf.difPos=0.7426 ;K
f.difDifPos= 0.0977;break;
case  89 :Kf.angle=-3.3375 ; Kf.difAngle=-0.5682 ; Kf.pos=1.7467 ; Kf.difPos=0.7397 ;K
f.difDifPos= 0.0983;break;
case  88 :Kf.angle=-3.3200 ; Kf.difAngle=-0.5726 ; Kf.pos=1.7439 ; Kf.difPos=0.7367 ;K
f.difDifPos= 0.0989;break;
*/
```

```
//case  87 :Kf.angle=-3.3021 ; Kf.difAngle=-0.5770 ; Kf.pos=1.7411 ; Kf.difPos=0.7337
;Kf.difDifPos= 0.0995;break;
case  86 :Kf.angle=-3.2838 ; Kf.difAngle=-0.5814 ; Kf.pos=1.7382 ; Kf.difPos=0.7306 ;K
f.difDifPos= 0.1001;break;
//case  85 :Kf.angle=-3.2651 ; Kf.difAngle=-0.5857 ; Kf.pos=1.7353 ; Kf.difPos=0.7274
;Kf.difDifPos= 0.1007;break;
case  84 :Kf.angle=-3.2460 ; Kf.difAngle=-0.5900 ; Kf.pos=1.7323 ; Kf.difPos=0.7242 ;K
f.difDifPos= 0.1013;break;
//case  83 :Kf.angle=-3.2264 ; Kf.difAngle=-0.5943 ; Kf.pos=1.7292 ; Kf.difPos=0.7209
;Kf.difDifPos= 0.1019;break;
case  82 :Kf.angle=-3.2065 ; Kf.difAngle=-0.5985 ; Kf.pos=1.7261 ; Kf.difPos=0.7175 ;K
f.difDifPos= 0.1026;break;
//case  81 :Kf.angle=-3.1861 ; Kf.difAngle=-0.6027 ; Kf.pos=1.7229 ; Kf.difPos=0.7141
;Kf.difDifPos= 0.1032;break;
case  80 :Kf.angle=-3.1652 ; Kf.difAngle=-0.6069 ; Kf.pos=1.7197 ; Kf.difPos=0.7106 ;K
f.difDifPos= 0.1039;break;
//case  79 :Kf.angle=-3.1438 ; Kf.difAngle=-0.6110 ; Kf.pos=1.7164 ; Kf.difPos=0.7070
;Kf.difDifPos= 0.1046;break;
case  78 :Kf.angle=-3.1220 ; Kf.difAngle=-0.6151 ; Kf.pos=1.7130 ; Kf.difPos=0.7033 ;K
f.difDifPos= 0.1053;break;
//case  77 :Kf.angle=-3.0996 ; Kf.difAngle=-0.6192 ; Kf.pos=1.7095 ; Kf.difPos=0.6995
;Kf.difDifPos= 0.1060;break;
case  76 :Kf.angle=-3.0767 ; Kf.difAngle=-0.6232 ; Kf.pos=1.7060 ; Kf.difPos=0.6956 ;K
f.difDifPos= 0.1067;break;
//case  75 :Kf.angle=-3.0533 ; Kf.difAngle=-0.6273 ; Kf.pos=1.7024 ; Kf.difPos=0.6917
;Kf.difDifPos= 0.1075;break;
case  74 :Kf.angle=-3.0293 ; Kf.difAngle=-0.6312 ; Kf.pos=1.6987 ; Kf.difPos=0.6877 ;K
f.difDifPos= 0.1082;break;
//case  73 :Kf.angle=-3.0047 ; Kf.difAngle=-0.6351 ; Kf.pos=1.6949 ; Kf.difPos=0.6835
;Kf.difDifPos= 0.1090;break;
case  72 :Kf.angle=-2.9795 ; Kf.difAngle=-0.6390 ; Kf.pos=1.6910 ; Kf.difPos=0.6793 ;K
f.difDifPos= 0.1098;break;
```

```
//case  71 :Kf.angle=-2.9537 ; Kf.difAngle=-0.6429 ; Kf.pos=1.6870 ; Kf.difPos=0.6750
;Kf.difDifPos= 0.1106;break;
case  70 :Kf.angle=-2.9273 ; Kf.difAngle=-0.6467 ; Kf.pos=1.6830 ; Kf.difPos=0.6705 ;K
f.difDifPos= 0.1115;break;
//case  69 :Kf.angle=-2.9002 ; Kf.difAngle=-0.6504 ; Kf.pos=1.6788 ; Kf.difPos=0.6660
;Kf.difDifPos= 0.1123;break;
case  68 :Kf.angle=-2.8724 ; Kf.difAngle=-0.6541 ; Kf.pos=1.6745 , Kf.difPos=0.6613 ;K
f.difDifPos= 0.1132;break;
//case  67 :Kf.angle=-2.8438 ; Kf.difAngle=-0.6578 ; Kf.pos=1.6702 ; Kf.difPos=0.6565
;Kf.difDifPos= 0.1141;break;
case  66 :Kf.angle=-2.8146 ; Kf.difAngle=-0.6614 ; Kf.pos=1.6657 ; Kf.difPos=0.6516 ;K
f.difDifPos= 0.1150;break;
//case  65 :Kf.angle=-2.7845 ; Kf.difAngle=-0.6649 ; Kf.pos=1.6611 ; Kf.difPos=0.6466
;Kf.difDifPos= 0.1160;break;
case  64 :Kf.angle=-2.7537 ; Kf.difAngle=-0.6684 ; Kf.pos=1.6564 ; Kf.difPos=0.6414 ;K
f.difDifPos= 0.1169;break;
//case  63 :Kf.angle=-2.7220 ; Kf.difAngle=-0.6719 ; Kf.pos=1.6515 ; Kf.difPos=0.6361
;Kf.difDifPos= 0.1179;break;
case  62 :Kf.angle=-2.6894 ; Kf.difAngle=-0.6753 ; Kf.pos=1.6466 ; Kf.difPos=0.6307 ;K
f.difDifPos= 0.1189;break;
//case  61 :Kf.angle=-2.6560 ; Kf.difAngle=-0.6786 ; Kf.pos=1.6415 ; Kf.difPos=0.6251
;Kf.difDifPos= 0.1200;break;
case  60 :Kf.angle=-2.6216 ; Kf.difAngle=-0.6819 ; Kf.pos=1.6362 ; Kf.difPos=0.6193 ;K
f.difDifPos= 0.1211;break;
/*
case  59 :Kf.angle=-2.5862 ; Kf.difAngle=-0.6851 ; Kf.pos=1.6309 ; Kf.difPos=0.6134 ;K
f.difDifPos= 0.1222;break;
case  58 :Kf.angle=-2.5498 ; Kf.difAngle=-0.6882 ; Kf.pos=1.6253 ; Kf.difPos=0.6073 ;K
f.difDifPos= 0.1233;break;
case  57 :Kf.angle=-2.5124 ; Kf.difAngle=-0.6913 ; Kf.pos=1.6196 ; Kf.difPos=0.6011 ;K
f.difDifPos= 0.1245;break;
case  56 :Kf.angle=-2.4738 ; Kf.difAngle=-0.6943 ; Kf.pos=1.6138 ; Kf.difPos=0.5946 ;K
f.difDifPos= 0.1257;break;
```

```
case  55 :Kf.angle=-2.4341 ; Kf.difAngle=-0.6972 ; Kf.pos=1.6077 ; Kf.difPos=0.5880 ;K
f.difDifPos= 0.1269;break;
case  54 :Kf.angle=-2.3932 ; Kf.difAngle=-0.7000 ; Kf.pos=1.6015 ; Kf.difPos=0.5811 ;K
f.difDifPos= 0.1282;break;
case  53 :Kf.angle=-2.3510 ; Kf.difAngle=-0.7028 ; Kf.pos=1.5951 ; Kf.difPos=0.5741 ;K
f.difDifPos= 0.1295;break;
case  52 :Kf.angle=-2.3075 ; Kf.difAngle=-0.7055 ; Kf.pos=1.5885 ; Kf.difPos=0.5668 ;K
f.difDifPos= 0.1309;break;
case  51 :Kf.angle=-2.2626 ; Kf.difAngle=-0.7081 ; Kf.pos=1.5817 ; Kf.difPos=0.5593 ;K
f.difDifPos= 0.1323;break;
case  50 :Kf.angle=-2.2162 ; Kf.difAngle=-0.7106 ; Kf.pos=1.5746 ; Kf.difPos=0.5515 ;K
f.difDifPos= 0.1337;break;
case  49 :Kf.angle=-2.1683 ; Kf.difAngle=-0.7130 ; Kf.pos=1.5673 ; Kf.difPos=0.5435 ;K
f.difDifPos= 0.1352;break;
case  48 :Kf.angle=-2.1188 ; Kf.difAngle=-0.7153 ; Kf.pos=1.5598 ; Kf.difPos=0.5352 ;K
f.difDifPos= 0.1367;break;
case  47 :Kf.angle=-2.0677 ; Kf.difAngle=-0.7175 ; Kf.pos=1.5521 ; Kf.difPos=0.5267 ;K
f.difDifPos= 0.1383;break;
case  46 :Kf.angle=-2.0148 ; Kf.difAngle=-0.7196 ; Kf.pos=1.5440 ; Kf.difPos=0.5178 ;K
f.difDifPos= 0.1400;break;
case  45 :Kf.angle=-1.9600 ; Kf.difAngle=-0.7216 ; Kf.pos=1.5357 ; Kf.difPos=0.5086 ;K
f.difDifPos= 0.1417;break;
case  44 :Kf.angle=-1.9032 ; Kf.difAngle=-0.7235 ; Kf.pos=1.5270 ; Kf.difPos=0.4991 ;K
f.difDifPos= 0.1434;break;
case  43 :Kf.angle=-1.8444 ; Kf.difAngle=-0.7252 ; Kf.pos=1.5181 ; Kf.difPos=0.4893 ;K
f.difDifPos= 0.1453;break;
case  42 :Kf.angle=-1.7833 ; Kf.difAngle=-0.7268 ; Kf.pos=1.5088 ; Kf.difPos=0.4791 ;K
f.difDifPos= 0.1472;break;
case  41 :Kf.angle=-1.7200 ; Kf.difAngle=-0.7283 ; Kf.pos=1.4991 ; Kf.difPos=0.4684 ;K
f.difDifPos= 0.1491;break;
case  40 :Kf.angle=-1.6542 ; Kf.difAngle=-0.7297 ; Kf.pos=1.4891 ; Kf.difPos=0.4574 ;K
f.difDifPos= 0.1512;break;
```

```
        */
default : Kf.angle=-3.3200 ; Kf.difAngle=-0.5726 ; Kf.pos=1.7439 ; Kf.difPos=0.7367 ;K
f.difDifPos= 0.0989;
        }
}

else if (z==3)
{
        switch(Lrop){
 //K when down
/*
case 100 :Kf.angle=-0.8444 ; Kf.difAngle=-0.1081 ; Kf.pos=1.9605 ; Kf.difPos=0.5669 ;K
f.difDifPos= 0.0536;break;
case  99 :Kf.angle=-0.8421 ; Kf.difAngle=-0.1089 ; Kf.pos=1.9600 ; Kf.difPos=0.5665 ;K
f.difDifPos= 0.0537;break;
case  98 :Kf.angle=-0.8399 ; Kf.difAngle=-0.1096 ; Kf.pos=1.9595 ; Kf.difPos=0.5660 ;K
f.difDifPos= 0.0538;break;
case  97 :Kf.angle=-0.8375 ; Kf.difAngle=-0.1104 ; Kf.pos=1.9589 ; Kf.difPos=0.5656 ;K
f.difDifPos= 0.0540;break;
case  96 :Kf.angle=-0.8351 ; Kf.difAngle=-0.1111 ; Kf.pos=1.9584 ; Kf.difPos=0.5651 ;K
f.difDifPos= 0.0541;break;
case  95 :Kf.angle=-0.8326 ; Kf.difAngle=-0.1119 ; Kf.pos=1.9578 ; Kf.difPos=0.5646 ;K
f.difDifPos= 0.0542;break;
case  94 :Kf.angle=-0.8301 ; Kf.difAngle=-0.1126 ; Kf.pos=1.9573 ; Kf.difPos=0.5641 ;K
f.difDifPos= 0.0543;break;
case  93 :Kf.angle=-0.8275 ; Kf.difAngle=-0.1133 ; Kf.pos=1.9567 ; Kf.difPos=0.5636 ;K
f.difDifPos= 0.0544;break;
case  92 :Kf.angle=-0.8248 ; Kf.difAngle=-0.1140 ; Kf.pos=1.9561 ; Kf.difPos=0.5631 ;K
f.difDifPos= 0.0545;break;
case  91 :Kf.angle=-0.8221 ; Kf.difAngle=-0.1147 ; Kf.pos=1.9556 ; Kf.difPos=0.5625 ;K
f.difDifPos= 0.0546;break;
case  90 :Kf.angle=-0.8193 ; Kf.difAngle=-0.1153 ; Kf.pos=1.9550 ; Kf.difPos=0.5620 ;K
f.difDifPos= 0.0548;break;
```

```
case  89 :Kf.angle=-0.8164 ; Kf.difAngle=-0.1160 ; Kf.pos=1.9544 ; Kf.difPos=0.5615 ;K
f.difDifPos= 0.0549;break;
case  88 :Kf.angle=-0.8135 ; Kf.difAngle=-0.1166 ; Kf.pos=1.9538 ; Kf.difPos=0.5609 ;K
f.difDifPos= 0.0550;break;
*/
//case  87 :Kf.angle=-0.8105 ; Kf.difAngle=-0.1172 ; Kf.pos=1.9532 ; Kf.difPos=0.5603
;Kf.difDifPos= 0.0551;break;
case  86 :Kf.angle=-0.8074 ; Kf.difAngle=-0.1178 ; Kf.pos=1.9526 ; Kf.difPos=0.5597 ;K
f.difDifPos= 0.0553;break;
//case  85 :Kf.angle=-0.8042 ; Kf.difAngle=-0.1184 ; Kf.pos=1.9519 ; Kf.difPos=0.5591
;Kf.difDifPos= 0.0554;break;
case  84 .Kf.angle=-0.8010 ; Kf.difAngle=-0.1190 ; Kf.pos=1.9513 ; Kf.difPos=0.5585 ;K
f.difDifPos= 0.0555;break;
//case  83 :Kf.angle=-0.7977 ; Kf.difAngle=-0.1195 ; Kf.pos=1.9507 ; Kf.difPos=0.5579
;Kf.difDifPos= 0.0556;break;
case  82 :Kf.angle=-0.7943 ; Kf.difAngle=-0.1200 ; Kf.pos=1.9500 ; Kf.difPos=0.5573 ;K
f.difDifPos= 0.0558;break;
//case  81 :Kf.angle=-0.7908 ; Kf.difAngle=-0.1205 ; Kf.pos=1.9493 ; Kf.difPos=0.5566
;Kf.difDifPos= 0.0559;break;
case  80 :Kf.angle=-0.7872 ; Kf.difAngle=-0.1210 ; Kf.pos=1.9487 ; Kf.difPos=0.5560 ;K
f.difDifPos= 0.0560;break;
//case  79 :Kf.angle=-0.7836 ; Kf.difAngle=-0.1214 ; Kf.pos=1.9480 ; Kf.difPos=0.5553
;Kf.difDifPos= 0.0562;break;
case  78 :Kf.angle=-0.7798 ; Kf.difAngle=-0.1219 ; Kf.pos=1.9473 ; Kf.difPos=0.5546 ;K
f.difDifPos= 0.0563;break;
//case  77 :Kf.angle=-0.7760 ; Kf.difAngle=-0.1223 ; Kf.pos=1.9466 ; Kf.difPos=0.5539
;Kf.difDifPos= 0.0565;break;
case  76 :Kf.angle=-0.7721 ; Kf.difAngle=-0.1227 ; Kf.pos=1.9459 ; Kf.difPos=0.5532 ;K
f.difDifPos= 0.0566;break;
//case  75 :Kf.angle=-0.7681 ; Kf.difAngle=-0.1230 ; Kf.pos=1.9452 ; Kf.difPos=0.5525
;Kf.difDifPos= 0.0568;break;
case  74 :Kf.angle=-0.7639 ; Kf.difAngle=-0.1234 ; Kf.pos=1.9444 ; Kf.difPos=0.5517 ;K
f.difDifPos= 0.0569;break;
```

```
//case  73 :Kf.angle=-0.7597 ; Kf.difAngle=-0.1237 ; Kf.pos=1.9437 ; Kf.difPos=0.5509
;Kf.difDifPos= 0.0570;break;
case  72 :Kf.angle=-0.7554 ; Kf.difAngle=-0.1240 ; Kf.pos=1.9430 ; Kf.difPos=0.5502 ;K
f.difDifPos= 0.0572;break;
//case  71 :Kf.angle=-0.7510 ; Kf.difAngle=-0.1242 ; Kf.pos=1.9422 ; Kf.difPos=0.5494
;Kf.difDifPos= 0.0574;break;
case  70 :Kf.angle=-0.7464 ; Kf.difAngle=-0.1245 ; Kf.pos=1.9414 ; Kf.difPos=0.5486 ;K
f.difDifPos= 0.0575;break;
//case  69 :Kf.angle=-0.7418 ; Kf.difAngle=-0.1247 ; Kf.pos=1.9406 ; Kf.difPos=0.5477
;Kf.difDifPos= 0.0577;break;
case  68 :Kf.angle=-0.7370 ; Kf.difAngle=-0.1248 ; Kf.pos=1.9398 ; Kf.difPos=0.5469 ;K
f.difDifPos= 0.0578;break;
//case  67 :Kf.angle=-0.7322 ; Kf.difAngle=-0.1250 ; Kf.pos=1.9390 ; Kf.difPos=0.5460
;Kf.difDifPos= 0.0580;break;
case  66 :Kf.angle=-0.7272 ; Kf.difAngle=-0.1251 ; Kf.pos=1.9382 ; Kf.difPos=0.5451 ;K
f.difDifPos= 0.0582;break;
//case  65 :Kf.angle=-0.7221 ; Kf.difAngle=-0.1252 ; Kf.pos=1.9374 ; Kf.difPos=0.5442
;Kf.difDifPos= 0.0583;break;
case  64 :Kf.angle=-0.7168 ; Kf.difAngle=-0.1252 ; Kf.pos=1.9365 ; Kf.difPos=0.5433 ;K
f.difDifPos= 0.0585;break;
//case  63 :Kf.angle=-0.7115 ; Kf.difAngle=-0.1252 ; Kf.pos=1.9357 ; Kf.difPos=0.5424
;Kf.difDifPos= 0.0587;break;
case  62 :Kf.angle=-0.7060 ; Kf.difAngle=-0.1252 ; Kf.pos=1.9348 ; Kf.difPos=0.5414 ;K
f.difDifPos= 0.0588;break;
//case  61 :Kf.angle=-0.7003 ; Kf.difAngle=-0.1251 ; Kf.pos=1.9339 ; Kf.difPos=0.5404
;Kf.difDifPos= 0.0590;break;
case  60 :Kf.angle=-0.6946 ; Kf.difAngle=-0.1250 ; Kf.pos=1.9330 ; Kf.difPos=0.5394 ;K
f.difDifPos= 0.0592;break;
/*
case  59 :Kf.angle=-0.6887 ; Kf.difAngle=-0.1249 ; Kf.pos=1.9321 ; Kf.difPos=0.5384 ;K
f.difDifPos= 0.0594;break;
case  58 :Kf.angle=-0.6826 ; Kf.difAngle=-0.1247 ; Kf.pos=1.9312 ; Kf.difPos=0.5373 ;K
f.difDifPos= 0.0596;break;
```

```
case  57 :Kf.angle=-0.6764 ; Kf.difAngle=-0.1244 ; Kf.pos=1.9302 ; Kf.difPos=0.5363 ;K
f.difDifPos= 0.0597;break;
case  56 :Kf.angle=-0.6700 ; Kf.difAngle=-0.1242 ; Kf.pos=1.9293 ; Kf.difPos=0.5352 ;K
f.difDifPos= 0.0599;break;
case  55 :Kf.angle=-0.6635 ; Kf.difAngle=-0.1239 ; Kf.pos=1.9283 ; Kf.difPos=0.5341 ;K
f.difDifPos= 0.0601;break;
case  54 :Kf.angle=-0.6569 ; Kf.difAngle=-0.1235 ; Kf.pos=1.9273 ; Kf.difPos=0.5329 ;K
f.difDifPos= 0.0603;break;
case  53 :Kf.angle=-0.6500 ; Kf.difAngle=-0.1231 ; Kf.pos=1.9263 ; Kf.difPos=0.5318 ;K
f.difDifPos= 0.0605;break;
case  52 :Kf.angle=-0.6430 ; Kf.difAngle=-0.1226 ; Kf.pos=1.9253 ; Kf.difPos=0.5306 ;K
f.difDifPos= 0.0607;break;
case  51 :Kf.angle=-0.6359 ; Kf.difAngle=-0.1221 ; Kf.pos=1.9243 ; Kf.difPos=0.5293 ;K
f.difDifPos= 0.0609;break;
case  50 :Kf.angle=-0.6285 ; Kf.difAngle=-0.1216 ; Kf.pos=1.9233 ; Kf.difPos=0.5281 ;K
f.difDifPos= 0.0611;break;
case  49 :Kf.angle=-0.6210 ; Kf.difAngle=-0.1209 ; Kf.pos=1.9222 ; Kf.difPos=0.5268 ;K
f.difDifPos= 0.0613;break;
case  48 :Kf.angle=-0.6133 ; Kf.difAngle=-0.1203 ; Kf.pos=1.9211 ; Kf.difPos=0.5255 ;K
f.difDifPos= 0.0615;break;
case  47 :Kf.angle=-0.6054 ; Kf.difAngle=-0.1195 ; Kf.pos=1.9201 ; Kf.difPos=0.5242 ;K
f.difDifPos= 0.0618;break;
case  46 :Kf.angle=-0.5973 ; Kf.difAngle=-0.1188 ; Kf.pos=1.9190 ; Kf.difPos=0.5228 ;K
f.difDifPos= 0.0620;break;
case  45 :Kf.angle=-0.5890 ; Kf.difAngle=-0.1179 ; Kf.pos=1.9178 ; Kf.difPos=0.5215 ;K
f.difDifPos= 0.0622;break;
case  44 :Kf.angle=-0.5805 ; Kf.difAngle=-0.1170 ; Kf.pos=1.9167 ; Kf.difPos=0.5200 ;K
f.difDifPos= 0.0624;break;
case  43 :Kf.angle=-0.5718 ; Kf.difAngle=-0.1160 ; Kf.pos=1.9156 ; Kf.difPos=0.5186 ;K
f.difDifPos= 0.0626;break;
case  42 :Kf.angle=-0.5629 ; Kf.difAngle=-0.1150 ; Kf.pos=1.9144 ; Kf.difPos=0.5171 ;K
f.difDifPos= 0.0629;break;
```

```
case  41 :Kf.angle=-0.5538 ; Kf.difAngle=-0.1139 ; Kf.pos=1.9132 ; Kf.difPos=0.5156 ;K
f.difDifPos= 0.0631;break;
case  40 :Kf.angle=-0.5444 ; Kf.difAngle=-0.1127 ; Kf.pos=1.9120 ; Kf.difPos=0.5141 ;K
f.difDifPos= 0.0633;break;
*/
default :Kf.angle=-0.8135 ; Kf.difAngle=-0.1166 ; Kf pos=1.9538 ; Kf.difPos=0.5609 ;Kf
.difDifPos= 0.0550;
        }
}  ;

else
{
        switch(Lrop){
// K when stop
/*
case 100 :Kf.angle=-1.7484 ; Kf.difAngle=-0.2400 ; Kf.pos=1.8939 ; Kf.difPos=0.6434 ;K
f.difDifPos= 0.0675;break;
case  99 :Kf.angle=-1.7428 ; Kf.difAngle=-0.2421 ; Kf.pos=1.8928 ; Kf.difPos=0.6423 ;K
f.difDifPos= 0.0678;break;
case  98 :Kf.angle=-1.7370 ; Kf.difAngle=-0.2442 ; Kf.pos=1.8916 ; Kf.difPos=0.6412 ;K
f.difDifPos= 0.0680;break;
case  97 :Kf.angle=-1.7311 ; Kf.difAngle=-0.2462 ; Kf.pos=1.8904 ; Kf.difPos=0.6401 ;K
f.difDifPos= 0.0682;break;
case  96 :Kf.angle=-1.7251 ; Kf.difAngle=-0.2482 ; Kf.pos=1.8892 ; Kf.difPos=0.6389 ;K
f.difDifPos= 0.0685;break;
case  95 :Kf.angle=-1.7189 ; Kf.difAngle=-0.2502 ; Kf.pos=1.8880 ; Kf.difPos=0.6378 ;K
f.difDifPos= 0.0687;break;
case  94 :Kf.angle=-1.7126 ; Kf.difAngle=-0.2522 ; Kf.pos=1.8867 ; Kf.difPos=0.6366 ;K
f.difDifPos= 0.0690;break;
case  93 :Kf.angle=-1.7061 ; Kf.difAngle=-0.2542 ; Kf.pos=1.8855 ; Kf.difPos=0.6354 ;K
f.difDifPos= 0.0693;break;
case  92 :Kf.angle=-1.6995 ; Kf.difAngle=-0.2561 ; Kf.pos=1.8842 ; Kf.difPos=0.6342 ;K
f.difDifPos= 0.0695;break;
```

```
case  91 :Kf.angle=-1.6927 ; Kf.difAngle=-0.2581 ; Kf.pos=1.8829 ; Kf.difPos=0.6329 ;K
f.difDifPos= 0.0698;break;
case  90 :Kf.angle=-1.6858 ; Kf.difAngle=-0.2600 ; Kf.pos=1.8815 ; Kf.difPos=0.6316 ;K
f.difDifPos= 0.0701;break;
case  89 :Kf.angle=-1.6787 ; Kf.difAngle=-0.2619 ; Kf.pos=1.8802 ; Kf.difPos=0.6303 ;K
f.difDifPos= 0.0704;break;
case  88 :Kf.angle=-1.6715 ; Kf.difAngle=-0.2637 ; Kf.pos=1.8788 ; Kf.difPos=0.6290 ;K
f.difDifPos= 0.0706;break;
*/
case  87 :Kf.angle=-1.6641 ; Kf.difAngle=-0.2656 ; Kf.pos=1.8774 ; Kf.difPos=0.6276 ;K
f.difDifPos= 0.0709;break;
case  86 :Kf.angle=-1.6565 ; Kf.difAngle=-0.2674 ; Kf.pos=1.8759 ; Kf.difPos=0.6262 ;K
f.difDifPos= 0.0712;break;
case  85 :Kf.angle=-1.6487 ; Kf.difAngle=-0.2691 ; Kf.pos=1.8745 ; Kf.difPos=0.6248 ;K
f.difDifPos= 0.0715;break;
case  84 :Kf.angle=-1.6407 ; Kf.difAngle=-0.2709 ; Kf.pos=1.8730 ; Kf.difPos=0.6233 ;K
f.difDifPos= 0.0718;break;
case  83 :Kf.angle=-1.6326 ; Kf.difAngle=-0.2726 ; Kf.pos=1.8715 ; Kf.difPos=0.6218 ;K
f.difDifPos= 0.0721;break;
case  82 :Kf.angle=-1.6242 ; Kf.difAngle=-0.2743 ; Kf.pos=1.8699 ; Kf.difPos=0.6203 ;K
f.difDifPos= 0.0724;break;
case  81 :Kf.angle=-1.6157 ; Kf.difAngle=-0.2760 ; Kf.pos=1.8684 ; Kf.difPos=0.6187 ;K
f.difDifPos= 0.0728;break;
case  80 :Kf.angle=-1.6069 ; Kf.difAngle=-0.2777 ; Kf.pos=1.8668 ; Kf.difPos=0.6172 ;K
f.difDifPos= 0.0731;break;
case  79 :Kf.angle=-1.5980 ; Kf.difAngle=-0.2793 ; Kf.pos=1.8651 ; Kf.difPos=0.6155 ;K
f.difDifPos= 0.0734;break;
case  78 :Kf.angle=-1.5888 ; Kf.difAngle=-0.2809 ; Kf.pos=1.8635 ; Kf.difPos=0.6139 ;K
f.difDifPos= 0.0738;break;
case  77 :Kf.angle=-1.5794 ; Kf.difAngle=-0.2824 ; Kf.pos=1.8618 ; Kf.difPos=0.6122 ;K
f.difDifPos= 0.0741;break;
case  76 :Kf.angle=-1.5697 ; Kf.difAngle=-0.2839 ; Kf.pos=1.8601 ; Kf.difPos=0.6104 ;K
f.difDifPos= 0.0745;break;
```

```
case  75 :Kf.angle=-1.5598 ; Kf.difAngle=-0.2854 ; Kf.pos=1.8583 ; Kf.difPos=0.6087 ;K
f.difDifPos= 0.0748;break;
default : Kf.angle=-1.6715 ; Kf.difAngle=-0.2637 ; Kf.pos=1.8788 ; Kf.difPos=0.6290 ;K
f.difDifPos = 0.0706;
//default is Kf at 88
        }
}

        /*subtitute the amount of K */

        f2=realDifAngle[k]*Kf.difAngle;          //use for LQR
        f3=realDifPos[k]*Kf.difPos;              //use for LQR
        f4=realDifDifPos[k]*Kf.difDifPos;        //use for LQR

        controlLaw[k]=(Kf.angle*errorAngle[k])+(Kf.pos*errorPos[k])-(f2+f3+f4);

                    /* For LQR */


        VELOCITY_OUT(controlLaw[k]);


        UPDATE_GRAPH(scale,realPos[k],ropPos[k],stopPos,realAngle[k],samplingTime);
       //NEW GRAPH
        samplingTime=samplingTime+info.sampling;


        zz[k]=z;
        yy[k]=Kf.angle;
        }while(samplingTime<maxTime);
        outp(ADD_RELAY,STOP_UP);
        outp(ADD_RELAY,STOP_UP);
```

```
        VELOCITY_OUT(0);

        CLOSE_GRAPHIC();                //close graphic mode
        clrscr();

        end=clock();                    //read clock when end routine
        time=(long)(((end-startProcess)/CLK_TCK)*1000000);
        gotoxy(X-6,Y);
        printf("\aThis Process use time = %10.0ld microsec",time);

        datafile=fopen(craneFile,"wt+");        //open file to write data
        gotoxy(X-6,Y+2);
        printf("Wait for writing data to file.");

        /* Write data to file */

        for (k=0;k<MAXDATA;k++)
        {
        fprintf(datafile,"%3.0d %2.8f %2.8f %2.8f %2.8f %2.8f %2.8f %2.8f %2.8f
\n",zz[k],yy[k],ropPos[k],controlLaw[k],errorPos[k],filAngle[k],realDifAngle[k],realPo
s[k],realDifPos[k],realDifDifPos[k]);
        }

        gotoxy(15,22);
        printf("Writing data to file is ready.\n\a");
        fclose(datafile);               //close file
        gotoxy(15,24);
        printf("Press any key to quit program.");

        getch();

}
```

# ADC6.H

```
/* SubProgram 14 bit Analog to Digital Conversion (in PCL-714 Broad)

   Last Update  20 MAR 1996
*/

#include <conio.h>
#include <math.h>
#include <time.h>

/* Define Constant Value */

#define BASE 0x220                  //base address of 8253
#define CH 12                       //old is ch 10 channel of ADC
#define TRIGGER 1
#define PACER 2



typedef struct{
        char high;
        char low;
        }word;                      //1 word = high byte + low byte

typedef struct{
        //word c0;                  //counter0
        word c1;                    //counter1
        word c2;                    //counter2
        }counterRegister;           //define counter register data type


counterRegister COUNTER_REGISTER(unsigned long period);
void SET_COUNTER(int base,counterRegister counter);
void TRIG(int mode,int base,int channel,counterRegister period,unsigned int k);
word READ_ADC(int base);
word READ_ADC1(int base);
counterRegister READ_COUNTER(int base);

/* Function */

counterRegister COUNTER_REGISTER(unsigned long period)
{
        unsigned int high=0,low=2;
```

```
        counterRegister counter;

  /*
     sampling period = (HB1*256+LB1)*(HB2*256+LB2)*0.5 microseconds

        assign high=(HB1*256+LB1)
        assign low=(HB2*256+LB2)
        counter.c1.high=HB1
        counter.c1.low=LB1
        counter.c2.high=HB2
        counter.c2.low=LB2

  */

        low=(int)sqrt(period);                  //define low word
        //low=10000;
        high=(period*2)/low;                    //compute high word
        counter.c1.high=high/256;       //high byte of counter2 of 8253
        counter.c1.low=high%256;        //low byte of counter2 of 8253
        counter.c2.high=low/256;        //high byte of counter1 of 8253
        counter.c2.low=low%256;         //low byte of counter1 of 8253

        return counter;                 //return value of counter register

}

void SET_COUNTER(int base,counterRegister counter)
{

        /* address base+0 use to read or write LSB or MSB of counter0
           address base+1 use to read or write LSB or MSB of counter1
           address base+2 use to read or write LSB or MSB of counter2
           address base+3 use to write CONTROL BYTE to 8253
        */

        outp(base+3,0x74);              //load control byte to counter1 of 8253
        outp(base+1,counter.c1.low);
        outp(base+1,counter.c1.high);
        outp(base+3,0xB4);              //load control byte to counter2 of 8253
        outp(base+2,counter.c2.low);
        outp(base+2,counter.c2.high);
        //return 1;
```

```
}

counterRegister READ_COUNTER(int base)
{

        counterRegister count;

    /* read data from counter */
        outp(base+3,64);                //latch data from counter1
        count.c1.low=inp(base+1);
        count.c1.high=inp(base+1);
        outp(base+3,192);               //latch data from counter2
        count.c2.low=inp(base+2);
        count.c2.high=inp(base+2);

        //time=(count.c1.high*256+count.c1.low)*(count.c2.high*256+count.c2.low);

    /* write counter data to screen*/
        gotoxy(20,15);
        printf("COUNTER1 = %d ",count.c1.low); //test
        gotoxy(20,17);
        printf("COUNTER1 = %d ",count.c1.high); //test
        gotoxy(20,19);
        printf("COUNTER2 = %d ",count.c2.low); //test
        gotoxy(20,21);
        printf("COUNTER2 = %d ",count.c2.high); //test

        return count;

}


word READ_ADC(int base)
{

        word data;
        //int clip=240;         //254 bit 1st=0,252 bit 2nd=0,248 bit 3rd=0,240 bit 4th=0

        /* address base+4 use to read ADC low byte
                        or use to write DAC low byte
           address base+5 use to read ADC high byte
                        or use to write DAC high byte
```

```
*/
        //int i=1;

        do
        {   /* wait for complete conversion */
                data.high=inp(base+5);          //read high_byte data

            /* read counter data wait for sampling time */
            //      if (!(i%25000))
            //      {
            //              READ_COUNTER(base);
            //      }
            //      i++;

        }while(data.high>=64);              //if hi >=64 mean conversion is not complete
        data.low=inp(base+4);               //read low_byte data
        //data.low=data.low&clip;                       //don't care last 2 bit

        return data;                        //return reading data

}

void TRIG(int mode,int base,int channel,counterRegister period,unsigned int k)
{

        /* address base+10 use to write (3bit) to select the desired ADC channel
           address base+11 use to write (2bit) to select ADC trigger mode control
        */

        outp(base+10,channel);              //select channel
        //if ((mode==PACER)&&(k==1))
        //if (mode==PACER)
        //{
        //      SET_COUNTER(base,period);
        //}
        outp(base+11,mode);                 //select mode

        //return 1;

}
```

```
/* Program test PLOTING IN GRAPHICS MODE with array input */

#include <conio.h>
#include <stdlib.h>
#include <dos.h>

/* Use graphics.h to include graphics module */

#include<graphics.h>



#define LEFT_GRAPH 0
#define TOP_GRAPH 0
#define RIGHT_GRAPH getmaxx()
#define BOTTOM_GRAPH (getmaxy()-70)
#define LEFT_DESCRIP 0
#define TOP_DESCRIP (getmaxy()-69)
#define RIGHT_DESCRIP getmaxx()
#define BOTTOM_DESCRIP getmaxy()
#define CLIP_ON  1
#define CLIP_OFF 0

#define XMIN 60
#define XMAX (RIGHT_GRAPH-39)
#define YMIN (BOTTOM_GRAPH-40)
#define YMAX 39
#define X_RANGE (XMAX-XMIN)
#define Y_RANGE (YMIN-YMAX)

/*
  COLOR CODE
  1 = BLUE
  14= YELLOW
  15= WHITE
*/

typedef struct{
        float xMax;          //Max Time Operation (sec)
```

```
        float yMax;          //Set Point Position (cm)
        float sampling;      //Sampling Period  (sec)
        char *heading;       //Heading of Graph (string)
        char *xLabel;        //Label of X Axis  (string)
        char *yLabel;        //Label of Y Axis  (string)
}graphInfo;

typedef struct{
                                //float pix;
        float secPpix;
        int pixPsampling;
}scaleX;

typedef struct{
        float cmPpix;
        }scaleY;

typedef struct{
        scaleX x;
        scaleY y;
        }axisScale;


/* Prototype Function */

int OPEN_GRAPHIC(void);
int CLOSE_GRAPHIC(void);
void DRAW_AXIS(void);
void WRITE_HEADING(char *heading,char *xLabel,char *yLabel);
axisScale WRITE_SCALE(float xMax,float yMax,float sampling);
void DRAW_GRID(int xGrid,int yGrid);
void DRAW_GRAPH(void);
//void UPDATE_GRAPH(axisScale scale,float distance,float time);
void UPDATE_GRAPH(axisScale scale,float distance,float ropdistance,float stpos,float angle,float time);
axisScale INITIAL_GRAPH(graphInfo info);
void SETVIEW(int left,int top,int right,int bottom,int clip);
void DESCRIPTION(float maxTime,float maxDistance);

/* Declare Static Parameter (NEW) */
```

```
int oldTime=XMIN;
int oldCraneValue=369;
int oldLoadValue=369;

/* Funtion */

int OPEN_GRAPHIC(void)
{
        int gdrive=DETECT,gmode,errorcode;

        /* Chain graphics function to program */

        errorcode = registerbgidriver(EGAVGA_driver);
        if (errorcode<0) /* Have error */
        {
                printf("Graphics error:%s\n",grapherrormsg(errorcode));
                //return 0;                     //initial is fail
                exit(1);
        }

        /* Initialize graphics */

        initgraph(&gdrive,&gmode,"");

        /* Check error */

        errorcode = graphresult();
        if(errorcode != grOk) /* An error occurred */
        {
                printf("Error while opening \n");
                //return 0;                     //initial is fail
                exit(2);
        }
        return 1;                       //initial is success
}
```

```
int CLOSE_GRAPHIC(void)
{
        /* Wait key stroke and change to text mode */
        sound(800);
        delay(400);
        nosound();
        getch();
        closegraph();
        return 1;
}


void WRITE_HEADING(char *heading,char *xLabel,char *yLabel)
{
        /* Write Title and X,Y label */

        settextjustify(CENTER_TEXT,CENTER_TEXT);        //set text justify
        setcolor(13);                                   //set Title color
        settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);         //set text style of Title
        outtextxy((XMAX/2),(YMAX-15),heading);          //write Title
        setcolor(13);                                   //set text color
        settextstyle(DEFAULT_FONT,HORIZ_DIR,1);         //set text style of X label
        outtextxy((XMAX/2),(YMIN+20),xLabel);           //write X label
        settextstyle(DEFAULT_FONT,VERT_DIR,1);          //set text style of Y label
        outtextxy((XMIN-40),(YMIN/2),yLabel);           //write Y label

}
axisScale WRITE_SCALE(float xMax,float yMax,float sampling)
{

        char *buffer;
        axisScale scale;
        unsigned int x1,y1,setpoint;
        float i;
        float xAll,yAll;

        scale.x.secPpix=xMax/(X_RANGE-60);
```

```
scale.x.pixPsampling = sampling/scale.x.secPpix;  //pixels per sampling
scale.y.cmPpix = yMax/(Y_RANGE-40);

/* Draw Set Point Line */

setcolor(15);
setlinestyle(SOLID_LINE,0,1);
setpoint=YMIN-(yMax/scale.y.cmPpix);
line(XMIN+1,setpoint,XMAX,setpoint);

/* Write Scale On X Axis */

setcolor(11);
settextstyle(SMALL_FONT,HORIZ_DIR,1);        //set text style of X Scale
setusercharsize(1,0.5,1,0.5);                //make small text
i=1;
for (x1=0;x1<X_RANGE;x1++)
{
        i=((x1)*scale.x.secPpix);            //increase scale

        if (!(x1%50))                        //write scale every 10 pixel
        {
                sprintf(buffer,"%.2f",i);
                outtextxy(XMIN+x1-4,YMIN+5,buffer);
        }
}

/* Write Scale On Y Axis */

settextstyle(SMALL_FONT,HORIZ_DIR,1);        //set text style of Y Scale
                // settextstyle(GOTHIC_FONT,HORIZ_DIR,1);        //set text style of Y
Scale
//settextjustify(RIGHT_TEXT,CENTER_TEXT);
setusercharsize(1,0.5,1,0.5);                //make small text
i=1;
for (y1=1;y1<Y_RANGE;y1++)
{
        i=((y1)*scale.y.cmPpix);             //increase scale
```

```
                if (!(y1%10))                    //write scale every 10 pixel
                {
                        sprintf(buffer,"%.2f",i);
                        outtextxy(XMIN-30,YMIN-y1-3,buffer);
                }

        }
        return scale;

}

void DRAW_AXIS(void)
{

        /* Draw X,Y axis */

        setbkcolor(1);              //set back ground color
        setcolor(15);                    //set X-axis and Y-axis color to white
        setlinestyle(SOLID_LINE,0,1);  //set line sytle ,linestyle=SOLID_LINE,pattern=0,
thickness=normal
        //setbkcolor(7);                 //set black ground color to gray
        line(XMIN,YMIN,XMIN,YMAX);      //draw Y-axis
        line(XMIN,YMIN,XMAX,YMIN);  //draw X-axis

}


void DRAW_GRID(int xGrid,int yGrid)
{

        int x1,y1;

        /* Draw grid */

        setcolor(8);                //set grid color
        setlinestyle(DOTTED_LINE,0,1);  //set line sytle ,linestyle=SOLID_LINE,pattern=0,
thickness=normal
        for (x1=(XMIN+xGrid);x1<XMAX;x1+=xGrid)
        {
```

```
            line(x1,YMIN-1,x1,YMAX-1);
        }
        for(y1=(YMIN-yGrid);y1>YMAX;y1-=yGrid)
        {
            line(XMIN,y1,XMAX,y1);
        }

}

void DESCRIPTION(float maxTime,float maxDistance)
{
        int line=10;
        char buffer[20];


        setcolor(14);
        settextjustify(CENTER_TEXT,CENTER_TEXT);
        settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
        outtextxy(LEFT_DESCRIP+85,TOP_DESCRIP,"DESIRE DISTANCE = ");
        setcolor(12);
        sprintf(buffer,"%.4f m",maxDistance);
        outtextxy(LEFT_DESCRIP+200,TOP_DESCRIP,buffer);
        setcolor(14);
        outtextxy(LEFT_DESCRIP+85,TOP_DESCRIP+line," TIME OPERATING = ");
        setcolor(12);
        sprintf(buffer,"%.4f sec",maxTime);
        outtextxy(LEFT_DESCRIP+208,TOP_DESCRIP+line,buffer);
        setcolor(14);
        outtextxy(LEFT_DESCRIP+85,TOP_DESCRIP+(2*line),"       DISTANCE = ");

}

void DRAW_GRAPH(void)
{

        setcolor(20);                       //set response color to
        setfillstyle(EMPTY_FILL,1);     //set fill style
```

```
        setlinestyle(SOLID_LINE,0,1);   //set line sytle ,linestyle=SOLID_LINE,pattern=0,
thickness=normal
        moveto(XMIN,YMIN);              //set current point (CP) to origin of graph (XMIN,YMIN)

}

//void UPDATE_GRAPH(axisScale scale,float distance,float time)
void UPDATE_GRAPH(axisScale scale,float distance,float ropdistance,float stpos,float angle,float time)
{

        int line=10;                    //line of text (text heigth)
        char buffer[20];                //buffer of data
        //int xNew,xValue,yValue;
        int newTime,newCraneValue,newLoadValue;

        newTime=XMIN+(time/scale.x.secPpix);
        newCraneValue=YMIN-(distance/scale.y.cmPpix);
        newLoadValue=YMIN-(angle/scale.y.cmPpix);

        setcolor(4);                    //set color of graph
        moveto(oldTime,oldCraneValue);
        lineto(newTime,newCraneValue); //draw new graph from old point to new point
        setcolor(14);
        moveto(oldTime,oldLoadValue);
        lineto(newTime,newLoadValue);
        oldTime=newTime;
        oldCraneValue=newCraneValue;
        oldLoadValue=newLoadValue;

        setcolor(13);                   //set color of data
        settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
        //setfillstyle(EMPTY_FILL,1);
        bar(LEFT_DESCRIP+150,TOP_DESCRIP+(1.5*line),LEFT_DESCRIP+550,TOP_DESCRIP+
(2.5*line));   //clear area to write new data
        bar(LEFT_DESCRIP+350,TOP_DESCRIP+(0.5*line),LEFT_DESCRIP+550,TOP_DESCRIP+
(1.5*line));   //clear area to write new data
        sprintf(buffer,"%.4f m",distance);                      //write data to buffer
        outtextxy(LEFT_DESCRIP+200,TOP_DESCRIP+(2*line),buffer);    //write data from buffer to
monitor
```

```
setcolor(14);
outtextxy(LEFT_DESCRIP+385,TOP_DESCRIP+(2*line),"ROPE DISTANCE = ");
setcolor(13);
sprintf(buffer,"%.4f m",ropdistance);                    //write data to buffer
outtextxy(LEFT_DESCRIP+500,TOP_DESCRIP+(2*line),buffer);   //write data from buffer to monitor

sprintf(buffer,"%.4f m",stpos);                          //write data to buffer
outtextxy(LEFT_DESCRIP+500,TOP_DESCRIP+(1*line),buffer);   //write data from buffer to monitor

        //setviewport(0,0,getmaxx(),getmaxy(),CLIP_OFF);

}

//axisScale INITIAL_GRAPH(float xMax,float yMax,int xGrid,int yGrid)
axisScale INITIAL_GRAPH(graphInfo info)
{

        axisScale scale;
        int topGraph,leftGraph,bottomGraph,rightGraph;
        int i;


        OPEN_GRAPHIC();

        //SETVIEW(LEFT_GRAPH,TOP_GRAPH,RIGHT_GRAPH,BOTTOM_GRAPH,CLIP_OFF);

        DRAW_AXIS();

        WRITE_HEADING(info.heading,info.xLabel,info.yLabel);
        scale = WRITE_SCALE(info.xMax,info.yMax,info.sampling);
        //DRAW_GRID(xGrid,yGrid);
        DRAW_GRID(50,10);

//SETVIEW(LEFT_DESCRIP,TOP_DESCRIP,RIGHT_DESCRIP,BOTTOM_DESCRIP,CLIP_OFF);
        DESCRIPTION(info.xMax,info.yMax);
//SETVIEW(LEFT_GRAPH,TOP_GRAPH,RIGHT_GRAPH,BOTTOM_GRAPH,CLIP_OFF);
        DRAW_GRAPH();
        return scale;
}
```

```
// LM629 ADDRESS PORT
#define ADDR_COM        0x308   //address for write command and read status flag
#define ADDR_DATA       0x309   //address for read and write data
#define posConvertC2M   3.0041758E-5    //multiplier constant for convert
                                //encoder count to distance in m.
#define posConvertM2C   33287           //multiplier constant for convert
                                //distance in m to velocity in
                                //count/sampies
#define veloConvertC2M  1.34297E-6      //multiplier constant for convert
                                //count per sec to velocity m/s
                                //1/33287*341.333333333E-6*65536
#define veloConvertM2C  744617.585      //multiplier constant for convert
                                //velocity m/s to count per sec
                                //33287*341.333333333E-6*65536
#define veloCorrectFact 1.144165        //correct factor for convert to
                                //actial velocity
#define accConvertM2C   254.162         //multiplier constant for convert
                                //acceleration m/s^2 to count/sec^2
                                //33287*(341.333333333E-6)^2*65536
#define accConvertC2m   3.934498E-3     //multiplier constant for convert
                                //acceleration count/sec^2 to m/s^2
                                //1/33287*(341.333333333E-6)^2*65536

#define kpConvert   0.307               //kp=(256*40/33287)*Kp
#define kiConvert   0.027               //ki=(6553
#define kdConvert   901.25
#define kp          108                 //old kp=28.5  //propertion gain in metre
#define ki          707.279             //integration gain in metre
#define kd          3.751               //old kd=3.6 //derivative gain in metre
#define il          50000               //integration limit
#define filterCommand   0x000F          //kp,ki,kd well be loaded
#define InitMaxAccCommand 0x1820        //trajectory control word a 18 hex
                                //HB program forward direction
```

```
                                    //velocity mode, acceleration
                                    //will be loaded
#define maxVelocity     1.2             //in m/s
#define maxAcceleration 4               //in m/s^2
#define posCommand      0x002A          //position mode ACC VELO position will
                                    //be loaded and it 's absolute
#define forwVeloCommand 0x1808          //forward direction velocity will be
                                    //loaded
#define rewVeloCommand  0x0808          //reward direction velocity will be
                                    //loaded
#define stopMotorCommand 0x0200         //stop motor abruptly trajectory
                                    //command
#define resetBreakCommand 0x0000            //reset all interrupt
//command code for lm629 define
#define RESET       0x00    //reset the lm629 data n
#define DFH         0x02    //define home data n
#define SIP         0x03    //set index position data n
#define LPEI        0x1B    //load position error for interrupt data 2
#define MSKI        0x1C    //mask interrupts data 2
#define RSTI        0x1D    //reset interrupts data 2
#define LFIL        0x1E    //funtion load parameter of filter to lm629
#define UDF         0x04    //update filter parameter
#define LTRJ        0x1F    //load trajectory
#define STT         0x01    //start trajectory
#define SBPA        0x20    //set break point absolute
#define RDSIGS      0x0C    //read signal register
#define RDDP        0x08    //read desired position
#define RDRP        0x0A    //read real position
#define RDDV        0x07    //read desired velocity
#define RDRV        0x0B    //read real velocity


/* PROTOTYPE */
```

```
void WAITBUSY(void);            //wait if lm629 busies
void WRITE_COM(unsigned char);          //write command to lm629
void WRITE_DATA_BYTE(unsigned char);    //write data 1 byte to lm629
void WRITE_DATA_2BYTES(int);    //write data 2 bytes to lm629
void WRITE_DATA_4BYTES(long);   //write data 4 bytes to lm629
void WRITE_DATA_WORD(int);      //write word data ONLY FILTER DATA
void WRITE_DATA_2WORDS(long);   //write 2 words data ONLY TRAJECTORY DATA
unsigned char READ_DATA_BYTE(void);     //read data from lm629
int  READ_DATA_2BYTES(void);    //read 2 BYTEs data from lm629
long READ_DATA_4BYTES(void);    //read 4 BYTEs daata from lm629
void SET_BREAK_POINT(float);    //set break point
void INIT_LM629(void);          //initial lm629
void WRITE_FILTER(void);        //write filter parameter to lm629
void ACCELERATION_OUT(float);   //init maximum acceleration
void POSITION_OUT(float,float,float);       //send position command to lm629
void VELOCITY_OUT(float);       //send velocity command to lm629
void STOP_MOTOR_IF_REACH(void); //stop motor abruptly;
float READ_DESIRED_POS(void);   //read desired position
float READ_REAL_POS(void);      //read real position
float READ_DESIRED_VELO(void);  //read desired velocity
float READ_REAL_VELO(void);     //read real velosity

/* FUNCTION */

void WAITBUSY()
{
        while(0x01 & inp(ADDR_COM)); //wait util busy flag is zero
}

void WRITE_COM(unsigned char comm)
{
        WAITBUSY();
        outp(ADDR_COM,comm);
```

```
}

void WRITE_DATA_BYTE(unsigned char data)
{
        WAITBUSY();
        outp(ADDR_DATA,data);
}

void WRITE_DATA_2BYTES(int data)
{
        char *temp;
        temp=(char *)&data;
        WAITBUSY();
        outp(ADDR_DATA,*(temp+1));
        WAITBUSY();
        outp(ADDR_DATA,*(temp));
}

void WRITE_DATA_4BYTES(long data)
{
        char *temp;
        temp=(char *)&data;
        WAITBUSY();
        outp(ADDR_DATA,*(temp+3));
        WAITBUSY();
        outp(ADDR_DATA,*(temp+2));
        WAITBUSY();
        outp(ADDR_DATA,*(temp+1));
        WAITBUSY();
        outp(ADDR_DATA,*temp);
}

void WRITE_DATA_WORD(int data)
```

```
{
        char *temp;
        temp=(char *)&data;
        WAITBUSY();
        outp(ADDR_DATA,*(temp+1));
        outp(ADDR_DATA,*(temp));
}

void WRITE_DATA_2WORDS(long data)
{
        char *temp;
        temp=(char *)&data;
        WAITBUSY();
        outp(ADDR_DATA,*(temp+3));
        outp(ADDR_DATA,*(temp+2));
        WAITBUSY();
        outp(ADDR_DATA,*(temp+1));
        outp(ADDR_DATA,*temp);
}

unsigned char READ_DATA_BYTE(void)
{
        WAITBUSY();
        return inp(ADDR_DATA);
}

int READ_DATA_2BYTES(void)
{
        char temp1[1];
        int *temp2;
        WAITBUSY();
        temp1[1]=inp(ADDR_DATA);
        WAITBUSY();
```

```
        temp1[0]=inp(ADDR_DATA);
        temp2=(int *)temp1;
        return *temp2;
}

long READ_DATA_4BYTES(void)
{
        char temp1[3];
        long *temp2;
        WAITBUSY();
        temp1[3]=inp(ADDR_DATA);
        WAITBUSY();
        temp1[2]=inp(ADDR_DATA);
        WAITBUSY();
        temp1[1]=inp(ADDR_DATA);
        WAITBUSY();
        temp1[0]=inp(ADDR_DATA);
        temp2=(long *)temp1;
        return *temp2;
}

void SET_BREAK_POINT(float b_point)
{
        WRITE_COM(SBPA);                //set break point
        WRITE_DATA_4BYTES((long)(b_point*posConvertM2C));
}

void INIT_LM629(void)
{
        int status;
        clrscr();
        clrscr();
        printf("\n>>CONTROL CRANE PROGRAM NOW ACTIVE ");
```

```
        sound(440);
        delay(550);
        nosound();
        WRITE_COM(RESET);       //reset command code
        delay(500);

        printf("\n>>INITIAL LM629 NO.1 NOW PASS ");
        WRITE_COM(DFH);         //define home command
        WRITE_COM(SIP);         //set index position command
        printf("\n>>INITIAL LM629 NO.2 NOW PASS ");
}

void WRITE_FILTER(void)
{
        /*WRITE_COM(0X1E);      //COMMAND CODE
        WRITE_DATA_BYTE(0X00);  //CONTROL WORD & DERIVATIVE
        WRITE_DATA_BYTE(0X0F);
        WRITE_DATA_BYTE(0X00);      //KP
        WRITE_DATA_BYTE(0X39);
        WRITE_DATA_BYTE(0X00);  //KI
        WRITE_DATA_BYTE(0X07);
        WRITE_DATA_BYTE(0X00);  //KD
        WRITE_DATA_BYTE(0X00);
        WRITE_DATA_BYTE(0X30);  //IL
        WRITE_DATA_BYTE(0XFF);*/

        unsigned int kpPulse,kiPulse,kdPulse,ilPulse;
        WRITE_COM(LFIL);                //COMMAND CODE
        WRITE_DATA_WORD(filterCommand); //CONTROL WORD & DERIVATIVE

        kpPulse=(unsigned int)(kp*kpConvert);
        WRITE_DATA_WORD(kpPulse);               //kp parameter
```

```
        kiPulse=(unsigned int)(ki*kiConvert);
        WRITE_DATA_WORD(kiPulse);              //ki parameter

        kdPulse=(unsigned int)(kd*kdConvert);
        WRITE_DATA_WORD(kdPulse);              //kd parameter

        ilPulse=(unsigned int)il;
        WRITE_DATA_WORD(ilPulse);              //il parameter

        WRITE_COM(UDF);                        //update filter parameter
}

void ACCELERATION_OUT(float acc)
{
        WRITE_COM(LTRJ);                       //COMMAND CODE

        WRITE_DATA_WORD(InitMaxAccCommand);    //load trjectory control ward
                                               //velocity mode acceleration
                                               //will be loaded
        WRITE_DATA_2WORDS((long)(acc*accConvertM2C));
        WRITE_COM(STT);
}

void POSITION_OUT(float pos,float velo,float accel)
{
        WRITE_COM(LTRJ);
        WRITE_DATA_WORD(posCommand);
        WRITE_DATA_2WORDS((long)(accel*accConvertM2C));
        WRITE_DATA_2WORDS((long)(velo*veloConvertM2C));
        WRITE_DATA_2WORDS((long)(pos*posConvertM2C));
        WRITE_COM(STT);
}
```

```
void VELOCITY_OUT(float velocity)
{
        long velo;

        if (velocity>maxVelocity)
                velocity=maxVelocity;

        if (velocity>maxVelocity)
                velocity=maxVelocity;

        velo=(long)(velocity*veloCorrectFact*veloConvertM2C);
        if (velo>=0)                    //forward direction
                {

                WRITE_COM(LTRJ);                //load trajectory command code
                WRITE_DATA_WORD(forwVeloCommand);
                WRITE_DATA_2WORDS(velo);
                WRITE_COM(STT);
                }
        if (velo<0)                     //reward direction
                {
                velo=velo*(-1);

                WRITE_COM(LTRJ);                //load trajectory command code
                WRITE_DATA_WORD(rewVeloCommand);
                WRITE_DATA_2WORDS(velo);
                WRITE_COM(STT);
                }
}

void STOP_MOTOR_IF_REACH(void)
{
```

```
        if(0x40 & inp(ADDR_COM))                    //stop motor abruptly;
            {
            WRITE_COM(LTRJ);
                        WRITE_DATA_WORD(stopMotorCommand);
                        WRITE_COM(STT);
            WRITE_COM(RSTI);
            WRITE_DATA_2BYTES(resetBreakCommand);
            }
}

float READ_DESIRED_POS(void)
{
        WRITE_COM(RDDP);                    //COMMAND CODE
        return (float)READ_DATA_4BYTES()*posConvertC2M; //read desired
                                            // position
}

float READ_REAL_POS(void)                   //read read position
{
        WRITE_COM(RDRP);                    //COMMAND CODE
        return (float)READ_DATA_4BYTES()*posConvertC2M; //read real position
}

float READ_DESIRED_VELO(void)               //read desired velosity
{
        WRITE_COM(RDDV);                    //COMMAND CODE
        return (float)READ_DATA_4BYTES()*veloConvertC2M;//read desired
                                            //velosity
}

float READ_REAL_VELO(void)                  //read real velosity
{
        WRITE_COM(RDRV);                    //COMMAND CODE
```

```
    return (float)(READ_DATA_2BYTES()*veloConvertC2M*65535);//read real velosity
}
```

```
// LM629 OR 628 ADDRESS PORT  FOR COUNT
#define ADDR_COM2        0x306   //address for write command and read status flag
#define ADDR_DATA2       0x307  //address for read and write data
#define posConvertC2M   3.0041758E-5        //multiplier constant for convert
                                            //encoder count to distance in m.
#define posConvertM2C   33287             //multiplier constant for convert
                                            //distance in m to velocity in
                                            //count/samples
#define veloConvertC2M  1.34297E-6        //multiplier constant for convert
                                            //count per sec to velocity m/s
                                            //1/33287*341.333333333E-6*65536
#define veloConvertM2C  744617.585        //multiplier constant for convert
                                            //velocity m/s to count per sec
                                            //33287*341.333333333E-6*65536
#define veloCorrectFact 1.144165     //correct factor for convert to
                                            //actial velocity
#define accConvertM2C   254.162             //multiplier constant for convert
                                            //acceleration m/s^2 to count/sec^2
                                            //33287*(341.333333333E-6)^2*65536
#define accConvertC2m   3.934498E-3         //multiplier constant for convert
                                            //acceleration count/sec^2 to m/s^2
                                            //1/33287*(341.333333333E-6)^2*65536

#define ropConvertM2rm 0.1324432 //multiplier constant for convert
                                            //realPos (m) to ropPos (m)
                                            //we use this with big core with multiply old value by
                                            // 16/5
#define ropConvertRM2m 24.1632 //multiplier constant for convert
                                            //ropPos (m) to realPos (m) to use READ_REAL_POS2

#define kpConvert   0.307             //kp=(256*40/33287)*Kp
#define kiConvert   0.027              //ki=(6553
#define kdConvert   901.25

#define kp             528.5          //old kp=28.5  //propertion gain in metre
#define ki             745.7         //integration gain in metre
#define kd             3.6                                          //old kd=3.6 //
derivative gain in metre
```

```
#define il                    50000          //integration limit
#define filterCommand   0x000F       //kp,ki,kd well be loaded
#define InitMaxAccCommand 0x1820     //trajectory control word a 18 hex
                                     //HB program forward direction
                                     //velocity mode, acceleration
                                     //will be loaded
#define maxVelocity     1.2          //in m/s
#define maxAcceleration 4            //in m/s^2
#define posCommand      0x002A       //position mode ACC VELO position will
                                     //be loaded and it 's absolute
#define forwVeloCommand 0x1808       //forward direction velocity will be
                                     //loaded
#define rewVeloCommand  0x0808       //reward direction velocity will be
                                     //loaded
#define stopMotorCommand 0x0200      //stop motor abruptly trajectory
                                     //command
#define resetBreakCommand 0x0000     //reset all interrupt
//command code for lm629 define
#define RESET          0x00   //reset the lm629 data n
#define DFH            0x02   //define home data n
#define SIP            0x03   //set index position data n
#define LPEI           0x1B   //load position error for interrupt data 2
#define MSKI           0x1C   //mask interrupts data 2
#define RSTI           0x1D   //reset interrupts data 2
#define LFIL           0x1E   //funtion load parameter of filter to lm629
#define UDF            0x04   //update filter parameter
#define LTRJ           0x1F   //load trajectory
#define STT            0x01   //start trajectory
#define SBPA           0x20   //set break point absolute
#define RDSIGS         0x0C   //read signal register
#define RDDP           0x08   //read desired position
#define RDRP           0x0A   //read real position
#define RDDV           0x07   //read desired velocity
#define RDRV           0x0B   //read real velocity


/* PROTOTYPE */
void WAITBUSY2(void);          //wait if lm629 busies
```

```
void WRITE_COM2(unsigned char);          //write command to lm629
void WRITE_DATA_BYTE2(unsigned char);    //write data 1 byte to lm629
void WRITE_DATA_2BYTES2(int);    //write data 2 bytes to lm629
void WRITE_DATA_4BYTES2(long);   //write data 4 bytes to lm629
void WRITE_DATA_WORD2(int);      //write word data ONLY FILTER DATA
void WRITE_DATA_2WORDS2(long);   //write 2 words data ONLY TRAJECTORY DATA
unsigned char READ_DATA_BYTE2(void);     //read data from lm629
int  READ_DATA_2BYTES2(void);    //read 2 BYTEs data from lm629
long READ_DATA_4BYTES2(void);    //read 4 BYTEs daata from lm629
void SET_BREAK_POINT2(float);  //set break point
void INIT_LM6292(void);          //initial lm629
void WRITE_FILTER2(void);        //write filter parameter to lm629
void ACCELERATION_OUT2(float);           //init maximum acceleration
void POSITION_OUT2(float,float,float);       //send position command to lm629
void VELOCITY_OUT2(float);       //send velocity command to lm629
void STOP_MOTOR_IF_REACH2(void);     //stop motor abruptly;
float READ_DESIRED_POS2(void);//read desired position
float READ_REAL_POS2(void);      //read real position
float READ_DESIRED_VELO2(void);          //read desired velocity
float READ_REAL_VELO2(void);     //read real velosity

/* FUNCTION */

void WAITBUSY2()
{
        while(0x01 & inp(ADDR_COM2)); //wait util busy flag is zero
}

void WRITE_COM2(unsigned char comm)
{
        WAITBUSY2();
        outp(ADDR_COM2,comm);
}

void WRITE_DATA_BYTE2(unsigned char data)
{
        WAITBUSY2();
        outp(ADDR_DATA2,data);
```

```
}

void WRITE_DATA_2BYTES2(int data)
{
        char *temp;
        temp=(char *)&data;
        WAITBUSY2();
        outp(ADDR_DATA2,*(temp+1));
        WAITBUSY2();
        outp(ADDR_DATA2,*(temp));
}

void WRITE_DATA_4BYTES2(long data)
{
        char *temp;
        temp=(char *)&data;
        WAITBUSY2();
        outp(ADDR_DATA2,*(temp+3));
        WAITBUSY2();
        outp(ADDR_DATA2,*(temp+2));
        WAITBUSY2();
        outp(ADDR_DATA2,*(temp+1));
        WAITBUSY2();
        outp(ADDR_DATA2,*temp);
}

void WRITE_DATA_WORD2(int data)
{
        char *temp;
        temp=(char *)&data;
        WAITBUSY2();
        outp(ADDR_DATA2,*(temp+1));
        outp(ADDR_DATA2,*(temp));
}

void WRITE_DATA_2WORD2(long data)
{
        char *temp;
```

```
        temp=(char *)&data;
        WAITBUSY2();
        outp(ADDR_DATA2,*(temp+3));
        outp(ADDR_DATA2,*(temp+2));
        WAITBUSY2();
        outp(ADDR_DATA2,*(temp+1));
        outp(ADDR_DATA2,*temp);
}

unsigned char READ_DATA_BYTE2(void)
{
        WAITBUSY2();
        return inp(ADDR_DATA2);
}

int READ_DATA_2BYTES2(void)
{
        char temp1[1];
        int *temp2;
        WAITBUSY2();
        temp1[1]=inp(ADDR_DATA2);
        WAITBUSY2();
        temp1[0]=inp(ADDR_DATA2);
        temp2=(int *)temp1;
        return *temp2;
}

long READ_DATA_4BYTES2(void)
{
        char temp1[3];
        long *temp2;
        WAITBUSY2();
        temp1[3]=inp(ADDR_DATA2);
        WAITBUSY2();
        temp1[2]=inp(ADDR_DATA2);
        WAITBUSY2();
        temp1[1]=inp(ADDR_DATA2);
        WAITBUSY2();
```

```
        temp1[0]=inp(ADDR_DATA2);
        temp2=(long *)temp1;
        return *temp2;
}

void SET_BREAK_POINT2(float b_point)
{
        WRITE_COM2(SBPA);               //set break point
        WRITE_DATA_4BYTES2((long)(b_point*posConvertM2C));
}

void INIT_LM6292(void)
{
        int status;
        clrscr();
        clrscr();
        printf("\n>>CONTROL CRANE PROGRAM NOW ACTIVE ");
        sound(440);
        delay(550);
        nosound();
        WRITE_COM2(RESET);      //reset command code
        delay(500);

        printf("\n>>INITIAL LM629 NO.1 NOW PASS ");
        WRITE_COM2(DFH);        //define home command
        WRITE_COM2(SIP);        //set index position command
        printf("\n>>INITIAL LM629 NO.2 NOW PASS ");
}

void WRITE_FILTER2(void)
{
        unsigned int kpPulse,kiPulse,kdPulse,ilPulse;
        WRITE_COM2(LFIL);               //COMMAND CODE
        WRITE_DATA_WORD2(filterCommand); //CONTROL WORD & DERIVATIVE

        kpPulse=(unsigned int)(kp*kpConvert);
        WRITE_DATA_WORD2(kpPulse);              //kp parameter
```

```
        kiPulse=(unsigned int)(ki*kiConvert);
        WRITE_DATA_WORD2(kiPulse);              //ki parameter

        kdPulse=(unsigned int)(kd*kdConvert);
        WRITE_DATA_WORD2(kdPulse);              //kd parameter

        ilPulse=(unsigned int)il;
        WRITE_DATA_WORD2(ilPulse);              //il parameter

        WRITE_COM2(UDF);                //update filter parameter
}

void ACCELERATION_OUT2(float acc)
{
        WRITE_COM2(LTRJ);                       //COMMAND CODE

        WRITE_DATA_WORD2(InitMaxAccCommand);    //load trjectory control ward
                                                //velocity mode acceleration
                                                //will be loaded
        WRITE_DATA_2WORD2((long)(acc*accConvertM2C));
        WRITE_COM2(STT);
}

void POSITION_OUT2(float pos,float velo,float accel)
{
        WRITE_COM2(LTRJ);
        WRITE_DATA_WORD2(posCommand);
        WRITE_DATA_2WORD2((long)(accel*accConvertM2C));
        WRITE_DATA_2WORD2((long)(velo*veloConvertM2C));
        WRITE_DATA_2WORD2((long)(pos*posConvertM2C));
        WRITE_COM2(STT);
}

void VELOCITY_OUT2(float velocity)
{
        long velo;
```



```
        if (velocity>maxVelocity)
```

```
            velocity=maxVelocity;

        if (velocity>maxVelocity)
            velocity=maxVelocity;

        velo=(long)(velocity*veloCorrectFact*veloConvertM2C);
        if (velo>=0)                    //forward direction
            {

            WRITE_COM2(LTRJ);           //load trajectory command code
            WRITE_DATA_WORD2(forwVeloCommand);
            WRITE_DATA_2WORD2(velo);
            WRITE_COM2(STT);
            }
        if (velo<0)                     //reward direction
            {
            velo=velo*(-1);

            WRITE_COM2(LTRJ);           //load trajectory command code
            WRITE_DATA_WORD2(rewVeloCommand);
            WRITE_DATA_2WORD2(velo);
            WRITE_COM2(STT);
            }
}

void STOP_MOTOR_IF_REACH2(void)
{
        if(0x40 & inp(ADDR_COM2))               //stop motor abruptly;
                                {
                                WRITE_COM2(LTRJ);
                                WRITE_DATA_WORD2(stopMotorCommand);
                                WRITE_COM2(STT);
                                WRITE_COM2(RSTI);
                                WRITE_DATA_2BYTES2(resetBreakCommand);
                                }
}

float READ_DESIRED_POS2(void)
```

```
{
        WRITE_COM2(RDDP);                       //COMMAND CODE
        return (float)READ_DATA_4BYTES2()*posConvertC2M; //read desired
                                                        // position
}

float READ_REAL_POS2(void)                      //read read position
{
        WRITE_COM2(RDRP);                       //COMMAND CODE
        return (float)READ_DATA_4BYTES2()*posConvertC2M*ropConvertM2rm; //read real position
}

float READ_DESIRED_VELO2(void)                  //read desired velosity
{
        WRITE_COM2(RDDV);                       //COMMAND CODE
        return (float)READ_DATA_4BYTES2()*veloConvertC2M;//read desired
                                                        //velosity
}

float READ_REAL_VELO2(void)                     //read real velosity
{
        WRITE_COM2(RDRV);                       //COMMAND CODE
        return (float)(READ_DATA_2BYTES2()*veloConvertC2M*65535);//read real velosity
}
```

# หนังสืออ้างอิง

1. นิรุตม์ นาคสุข สุนิบูณ ตั้งวิญญู ,การควบคุมการเคลื่อนที่ของเครน,คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง,2539

2. รศ.วิพันธ์ ปรีชาพานิช,ระบบควบคุมออพติมัล,คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง,2532

3. ธันวา ศรีประโมง,การเขียนโปรแกรมภาษาซีสำหรับวิศวกรรม,มหาวิทยาลัยมหานคร ,2537

4. Frank L. Lewis, *Applied Optimal Control & Estimation*, Prentice-Hall International,Inc.,1992

5. Kasuhiko Okata , *Model Control Engineering* , Prentic-Hall International ,Inc ,1992.

6. Kasuhiko Okata , *Design Linear Control System with MATHLAB* , Prentic-Hall International ,Inc. ,1994.

7. Noriyuki Kornine , *Optimal Control of Anti-Sway for Overhead Traveling Crane* , Tokai University ,1995