การควบคุมการเคลื่อนที่ของเครน

CRANE MOVEMENT CONTROL

โดย

นายนิรุตต์ นาคสุข

นายสุนิบูณ ตั้งวิญญู

อาจารย์ที่ปรึกษา

รองศาสตราจารย์ ดร. จงกล งามวิวิทย์

อาจารย์ สุมิตร พนาอุดมทรัพย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมระบบควบคุม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2538

ปริญญานิพนธ์ปีการศึกษา 2538

ภาควิชาวิศวกรรมระบบควบคุม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การควบคุมการเคลื่อนที่ของเครน

CRANE MOVEMENT CONTROL

ผู้จัดทำ นายนิรุตต์ นาคสุข

นายสุนิบูณ ตั้งวิญญู

..... อาจารย์ที่ปรึกษา

(รองศาสตราจารย์ ดร. จงกล งามวิวิทย์)

..... อาจารย์ที่ปรึกษา

(อาจารย์ สุมิตร พนาอุดมทรัพย์)

การควบคุมการเคลื่อนที่ของเครน
CRANE MOVEMENT CONTROL

โดย
นายนิรุตต์ นาคสุข
รหัสประจำตัวนักศึกษา 35104222
นายสุนิบูณ ตั้งวิญญู
รหัสประจำตัวนักศึกษา 35104488

อาจารย์ที่ปรึกษา
รองศาสตราจารย์ ดร.จงกล งามวิวิทย์
อาจารย์ สุมิตร พนาอุดมทรัพย์

บทคัดย่อ

โครงงานการควบคุมการเคลื่อนที่ของเครน มีจุดมุ่งหมายเพื่อควบคุมรถเครนให้เคลื่อนที่ไปหยุดยังตำแหน่งที่ต้องการ ในขณะเดียวกันจะทำการควบคุมให้ภาระที่แขวนอยู่แกว่งน้อยที่สุด ซึ่งการออกแบบระบบควบคุมการเคลื่อนที่ของเครนประกอบด้วย 2 ขั้นตอน โดยในขั้นตอนแรกจะทำการหาความสัมพันธ์ของพารามิเตอร์ภายในระบบ แล้วทำการประมาณค่าโมเดลทางคณิตศาสตร์ของระบบโดยพิจารณาจากผลตอบสนองที่ได้จากการทดลอง ในขั้นที่สองจะทำการออกแบบระบบควบคุมโดยวิธี LQR และ LQG จากโมเดลทางคณิตศาสตร์ที่ได้ เพื่อแก้ปัญหาการแกว่งของภาระ ขณะที่รถเครนเคลื่อนที่ไปจนกระทั่งหยุด

ABSTRACT

The Crane Movement Control Project has main purpose to control the position of cart and the anti-swing of load in the same time. The design procedure is given by two step approaches. First, we determine the relation of parameters and identify a mathematical model by considering the experimental responses of the system. Second, we use the LQR and LQG method to control position of the crane.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยดีก็เพราะได้รับความเมตตาจาก รองศาสตราจารย์ ดร. จงกล งามวิวิทย์ อาจารย์ สุมิตร พนาอุดมทรัพย์ อาจารย์ที่ปรึกษา รองศาสตราจารย์ ดร. โยธิน เปรมปราณีรัชต์ รองศาสตราจารย์ ดร. สุเธียร เกียรติสุนทร ดร. สุธี ผู้เจริญชนะชัย รองศาสตราจารย์ ดร.วิพันธ์ ปรีชาพานิช ดร. นนทวัฒน์ จุลเดชะ ดร.วันชัย ริ้วรุจา อาจารย์ เกียรติวรรณ ทรงสัตย์ พี่ธิดาพร พี่พัฒพงษ์ พี่เทพจิตร พี่สมศักดิ์ ที่ได้ให้ความกรุณาแนะนำแก่ผู้จัดทำตลอดมา ผู้จัดทำขอกราบขอบพระคุณเป็นอย่างสูงมา ณ โอกาสนี้ด้วย และยังขอขอบคุณคณาจารย์ต่าง ๆ ที่ได้ประสิทธิประสาทวิชาความรู้แก่ผู้จัดทำ

ขอขอบคุณ บริษัท อีซูซุ(ประเทศไทย) จำกัด ที่ได้ให้ความอนุเคราะห์อุปกรณ์ที่ใช้ในระบบควบคุมการเคลื่อนที่ของเครน ซึ่งทำให้ปริญญานิพนธ์เรื่องนี้สำเร็จลงได้ด้วยดี

ขอกราบขอบพระคุณคุณพ่อคุณแม่ของผู้จัดทำที่ได้อุปการะผู้จัดทำและยังเป็นผู้ให้กำลังใจแก่ผู้จัดทำตลอดมาและต้องขอขอบคุณเพื่อน ๆ ที่ให้ความเป็นเพื่อนและให้กำลังใจแก่ผู้จัดทำ

ผู้จัดทำ
นายนิรุตต์ นาคสุข
นายสุนิบุณ ตั้งวิญญู

สารบัญ

บทที่1
บทนำ

1.1 จุดประสงค์ของการศึกษาและการควบคุมการเคลื่อนที่ของเครน

จุดประสงค์หลักในการควบคุมการเคลื่อนที่ของเครนมีเหตุผลดังจะกล่าวต่อไปนี้

1.1.1 ช่วยเพิ่มประสิทธิภาพในการทำงานโดยเฉลี่ย แม้ว่าผู้ปฏิบัติงานจะไม่มีความถนัด

1.1.2 ช่วยเพิ่มประสิทธิภาพการทำงานในสภาวะแวดล้อมที่ไม่เหมาะสม เช่น การที่ผู้ปฏิบัติงานที่ต้องทำงานในบริเวณที่มีอุณหภูมิสูง ในบริเวณที่มีกลิ่นเหม็น หรือในบริเวณที่มีทรรศนะวิสัยไม่ดี เป็นต้น

1.1.3 ช่วยลดค่าจ้างแรงงาน

1.1.4 สามารถประยุกต์ใช้งานกับกระบวนการผลิตในโรงงานอุตสาหกรรมได้อย่างมีประสิทธิภาพ

1.2 ประวัติการศึกษาการควบคุมเครน

ในปี คศ. 1979 Mita และ Kannai ได้พัฒนาวิธีการควบคุมการแกว่งของภาระโดยวิธี การควบคุมออพติมัล(Optimal Control) เพื่อให้ใช้เวลาในการควบคุมน้อยที่สุด

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

ในปี คศ. 1993 J. Shirai ได้ทำการพัฒนาวิธีการคำนวณแบบใหม่เพื่อใช้ในการควบคุมการแกว่งของภาระที่แขวนอยู่บนเครนที่ใช้ในการขนตู้คอนเทนเนอร์ภายในท่าเรือ

1.3 เนื้อหาที่จะกล่าวถึงในปริญญานิพนธ์ฉบับนี้

ในปริญญานิพนธ์ฉบับนี้จะประกอบไปด้วยเนื้อหาคร่าวๆดังนี้

1.การออกแบบโครงสร้างทางกลและวงจรอิเล็กทรอนิกส์ที่ใช้ในการวัดค่าสัญญาณต่างๆที่ใช้ในแบบจำลองเครน

2.การประมาณค่าโมเดลทางคณิตศาสตร์ของระบบเครน

3.การควบคุมการเคลื่อนที่ของเครนโดยการป้อนกลับสเตท (State Feedback) ซึ่งออกแบบค่าเกนป้อนกลับ(Feedback Gain) โดยวิธี Linear Quadratic Regulator (LQR)

4.การประมาณค่าสเตทโดยใช้ คาล์มานฟิลเตอร์ (Kalman Filter) และ การประมาณค่าสเตทจากการใช้ความสัมพันธ์ทางฟิสิกส์

5.การทดลองและผลการทดลอง

บทที่ 2
โครงสร้างโดยรวมของระบบ

2.1 หลักการในการควบคุมและบล็อกไดอะแกรมของระบบ

2.1.1 หลักการในการควบคุม

เนื่องจากการหมุนของมอเตอร์สามารถควบคุมระยะทางที่เคลื่อนที่ไปได้ของเครนและควบคุมมุมที่เกิดจากการแกว่งของภาระของเครน ดังนั้นเราจึงต้องควบคุมการหมุนของมอเตอร์ให้ได้ เพื่อจะทำให้เครนเคลื่อนที่ไปยังตำแหน่งที่ต้องการได้ และในขณะเดียวกันก็ทำให้เกิดการแกว่งของภาระของเครนมีค่าน้อยที่สุด

เราสามารถควบคุมการหมุนของมอเตอร์ได้จาก คอมพิวเตอร์ ชุดควบคุม และเซ็นเซอร์ (Sensor)ที่ทำงานร่วมกัน โดย เซ็นเซอร์จะทำการวัดสัญญาณป้อนกลับ (Feedback Signal) ค่าตำแหน่งของรถเครนและค่ามุมการแกว่ง คอมพิวเตอร์จะนำค่าที่ได้จากการวัดมาทำการคำนวณหาค่าสเตทต่างๆ และสัญญาณควบคุม (Control Signal) หรือสัญญาณความเร็วอ้างอิง (Velocity Reference Command) และทำการส่งสัญญาณควบคุมไป ยังชุดควบคุม เพื่อควบคุมให้เครนเคลื่อนที่ไปยังตำแหน่งที่ต้องการและมีมุมการแกว่งของภาระเป็นศูนย์ โดยไม่มีค่าผิดพลาดหรือมีค่าผิดพลาดน้อยที่สุด

2.1.2 บล็อกไดอะแกรมของระบบ

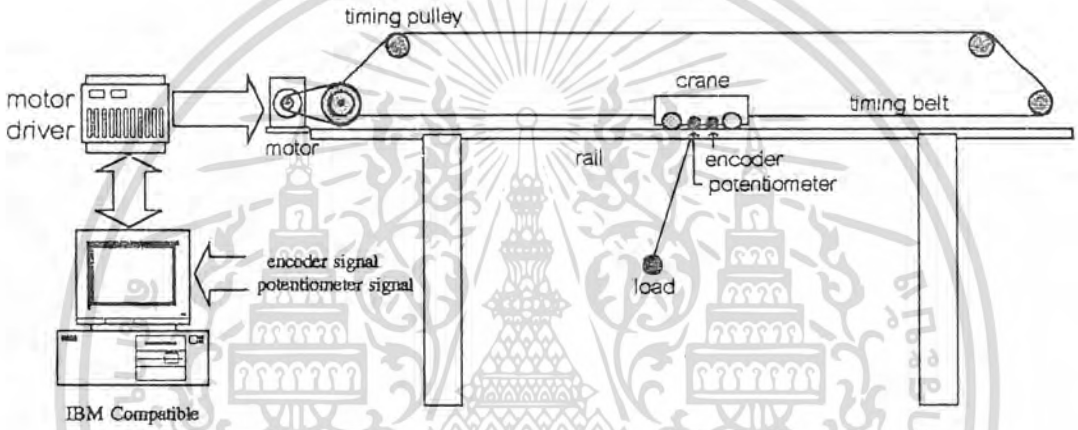
จากหัวข้อที่แล้ว สามารถเขียนบล็อกไดอะแกรมโดยรวมของระบบได้ดังนี้

รูปที่ 2.1 รูปแสดงบล็อกไดอะแกรมโดยรวมของระบบการควบคุมการเคลื่อนที่ของเครน

2.2 โครงสร้างและส่วนประกอบของแบบจำลองเครน

2.2.1 การออกแบบโครงสร้างทางกลของแบบจำลองเครน

แบบจำลองระบบเครนที่ออกแบบไว้ใช้รางที่ทำด้วยแสตนเลสมีระยะวิ่งประมาณ 2.1 เมตร โดยใช้ดีซีมอเตอร์เป็นตัวส่งกำลัง ส่งกำลังไปขับเคลื่อนรถเครนผ่านระบบส่งกำลัง เพื่อให้ได้แรงบิดสูงขึ้น 3 เท่า และให้มีค่าแบ็คแลช(back latch)และค่าสลิป(slip)น้อยที่สุด เพื่อให้ได้ความแม่นยำและรวดเร็วที่สุดในการควบคุมการเคลื่อนที่ของตัวเครน เราสามารถแสดงโครงสร้างและส่วนประกอบของแบบจำลองเครน ได้ดังรูปที่ 2.2ก



รูปที่2.2ก รูปแสดงโครงสร้างและส่วนประกอบของแบบจำลองเครน

คุณสมบัติของอุปกรณ์ต่างๆของแบบจำลองเครน

2.2.1.1 ตัวส่งกำลัง เราเลือกใช้ดีซีมอเตอร์แบบแม่เหล็กถาวรเป็นตัวส่งกำลังไปยังตัวเครน ซึ่งให้อัตราส่วนระหว่างกระแสอาร์เมเจอร์และแรงบิดจะมีค่าคงที่ ให้ความสัมพันธ์ระหว่างกระแสอาร์เมเจอร์ แรงบิดและความเร็ว เป็นเชิงเส้น จึงทำให้สามารถหาโมเดลได้ง่าย และ ไม่มีกำลังสูญเสียในฟิลด์ มีประสิทธิภาพสูงกว่า ให้ค่ากระแสอาร์เมเจอร์สูงกว่าและขนาดเล็กกว่าเมื่อเทียบกับดีซีมอเตอร์แบบฟิลด์เป็นขดลวดที่มีขนาดแรงม้าเท่ากัน

2.2.1.2 ระบบส่งกำลัง เลือกใช้ระบบส่งกำลังที่ประกอบด้วย

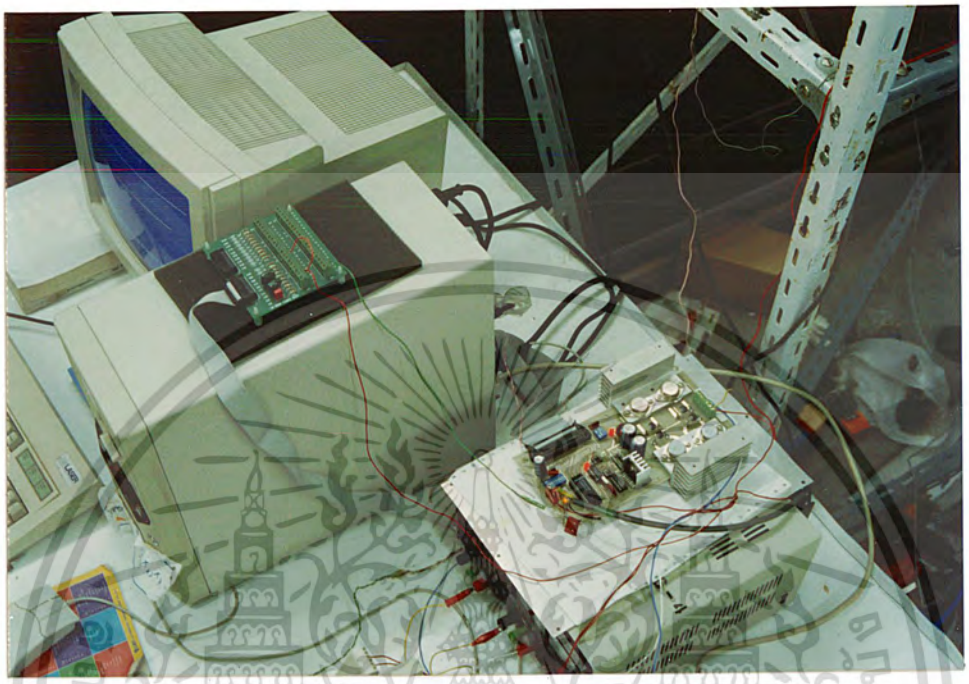
ก. ชุดทดรอบ ประกอบด้วยสายพานแบบไทม์มิ่งเบลท์ (timing belt) และไทม์มิ่งพูลเลย์ (timing pulley) โดยใช้ไทม์มิ่งพูลเลย์ 2 ขนาดที่มีอัตราส่วนของรัศมีของไทม์มิ่งพูลเลย์ที่ยึดอยู่กับมอเตอร์เป็น 1/3 ของไทม์มิ่งพูลเลย์ที่ยึดกับแกนหมุนของชุดส่งกำลังที่ส่งกำลังไปยังระบบ (ดังรูปที่2.3ก) เพื่อทดรอบมอเตอร์ให้ความเร็วลดลงแต่ให้แรงบิดเพิ่มขึ้น 3 เท่า การใช้ไทม์มิ่งเบลท์และไทม์มิ่งพูลเลย์นี้สามารถลดแบ็คแลชที่เกิดขึ้นในระบบทดรอบที่ใช้เฟืองเกียร์ได้

รูปที่ 2.2ข แสดงถึงระบบควบคุมการเคลื่อนที่ของเครนโดยรวม ประกอบด้วยโครงสร้างทางกล อุปกรณ์ตรวจวัดทางอิเล็กทรอนิกส์ และ คอมพิวเตอร์ที่ใช้ในการควบคุม

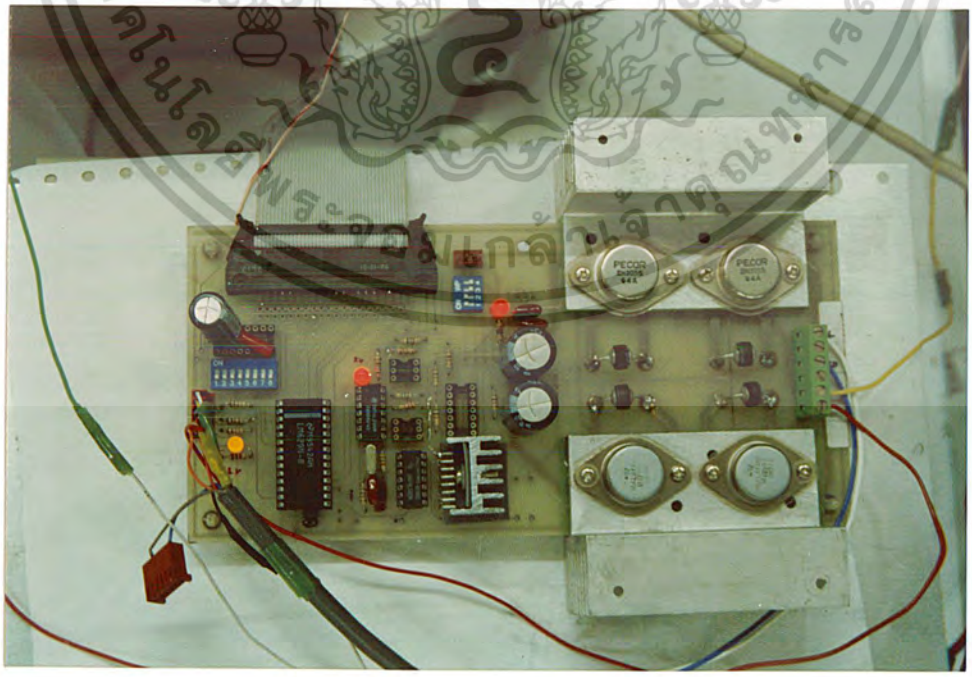
รูปที่ 2.2ค แสดงระบบส่งกำลังของระบบเครนที่ใช้ไทม์มิ่งเบลท์และไทม์มิ่งพูลเลย์ ในการทด รอบมอเตอร์ และส่งผ่านกำลังจากมอเตอร์มายังรถเครน

รูปที่ 2.2ง แสดงการวัดตำแหน่งของรถเครนโดยใช้เอนโคดเดอร์ซึ่งแกนต่ออยู่กับล้อยางที่วางให้สัมผัสกับรางสแตนเลส เมื่อรถเครนเคลื่อนที่ล้อยางนั้นก็จะหมุนไปบนรางทำให้รับรู้การเคลื่อนที่ของรถเครนได้

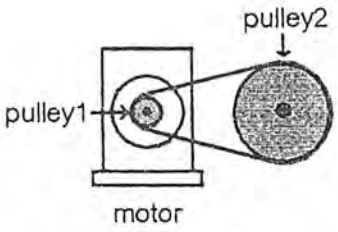
รูปที่ 2.2จ แสดงการใช้ โพเทนทิโอมิเตอร์ในการวัดมุมการแกว่งของภาระที่แขวนอยู่บนรถเครน



รูปที่ 2.2ฉ แสดงการเชื่อมต่อระหว่างชุดวงจรควบคุมความเร็วในลูปความเร็ว กับ คอมพิวเตอร์

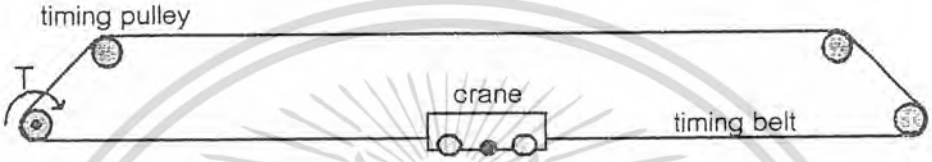


รูปที่ 2.2ช แสดงวงจรควบคุมความเร็วในลูปความเร็วซึ่งประกอบด้วย LM629 และ ชุดวงจรขับ



รูปที่2.3ก รูปแสดงการส่งกำลังในแบบที่มีการทดรอบของไทม์มิ่งพูลเลย์

ข. ชุดส่งกำลัง ประกอบด้วยสายพานแบบไทม์มิ่งเบลท์กับไทม์มิ่งพูลเลย์ ส่งกำลังไปยังตัวเครนโดยตรง(ดังรูปที่ 2.3ข) เพื่อลดสลิปที่เกิดในระบบที่ส่งกำลังโดยสลิงหรือสายพานธรรมดา



รูปที่ 2.3ข รูปแสดงการส่งกำลังในแบบที่ไม่มีการทดรอบของไทม์มิ่งพูลเลย์

2.3 ชุดควบคุมและเซ็นเซอร์

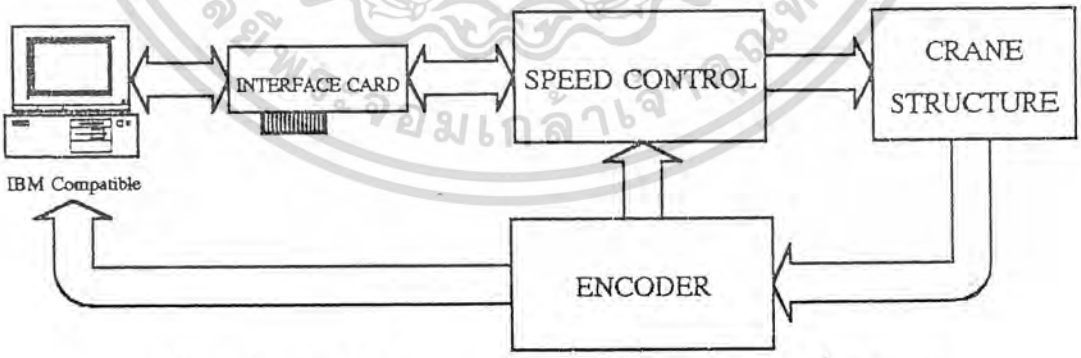
2.3.1 ชุดควบคุมการเคลื่อนที่ของเครน

สำหรับชุดควบคุมการเคลื่อนที่ของเครนนั้นจะประกอบไปด้วย

2.3.1.1 คอมพิวเตอร์ส่วนบุคคล (Personal Computer,PC)

2.3.1.2 วงจรเชื่อมต่อระหว่างคอมพิวเตอร์และชุดควบคุมความเร็ว (Interface Card)

2.3.1.3 ชุดควบคุมความเร็ว (Speed Control)



รูปที่2.4 รูปแสดงบล็อคไดอะแกรมของชุดควบคุมการเคลื่อนที่ของเครน

2.3.1.1 คอมพิวเตอร์ส่วนบุคคล

คอมพิวเตอร์ที่เลือกใช้เป็นแบบที่เข้ากันได้กับของบริษัท IBM เนื่องจากเป็นคอมพิวเตอร์ส่วนบุคคลที่หาได้ง่ายและนิยมใช้กันอย่างแพร่หลายในปัจจุบัน

คอมพิวเตอร์ที่ใช้นี้เป็นอุปกรณ์ที่ทำหน้าที่คำนวณหาค่าสัญญาณควบคุมหรือสัญญาณความเร็วอ้างอิงจากค่าสัญญาณป้อนกลับตำแหน่งของรถเครนกับค่าสัญญาณป้อนกลับมุมการแกว่ง และทำหน้าที่ติดต่อกับชุดควบคุมความเร็ว โดยทำการหาค่าสัญญาณควบคุมจากการนำค่าสัญญาณป้อนกลับมาทำการคำนวณหาค่าสัญญาณควบคุมที่ต้องการบนพื้นฐานของการป้อนกลับสเตท (State Feedback) ที่ออกแบบค่าเกนป้อนกลับ (Feedback Gain) โดยวิธี Linear Quadratic Regulator (ดูรายละเอียดในบทที่ 3) และทำการประมาณค่าสเตท (State Estimator) ที่ใช้ในการป้อนกลับจากค่าสัญญาณป้อนกลับที่ได้ โดยวิธีแรกใช้การคำนวณจากความสัมพันธ์ทางฟิสิกส์ (ดูรายละเอียดในบทที่ 3) และวิธีที่สองโดยออกแบบตัวสังเกตุสเตท (State Observer) จากวิธีคาล์มานฟิลเตอร์ (Kalman Filter) (ดูรายละเอียดในบทที่ 3) จากนั้นก็ทำการส่งสัญญาณควบคุมที่คำนวณได้ไปยังชุดควบคุมความเร็ว โดยติดต่อผ่านทางวงจรเชื่อมต่อที่ได้ออกแบบขึ้น

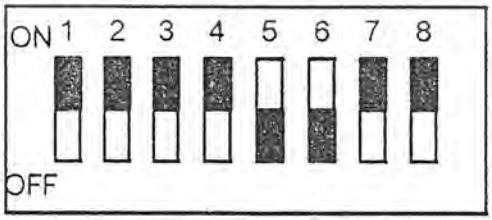
2.3.1.2 วงจรเชื่อมต่อระหว่างคอมพิวเตอร์ และวงจรควบคุมความเร็ว

วงจรเชื่อมต่อนี้ เป็นวงจรที่ออกแบบขึ้นเพื่อใช้ในระบบควบคุมการเคลื่อนที่ของเครน โดยวงจรนี้จะทำหน้าที่เป็นเหมือนพอร์ท (Port) และทำหน้าที่เป็นบัฟเฟอร์ของสัญญาณข้อมูล (Data Buffer) และบัฟเฟอร์ของสัญญาณควบคุมต่างๆ (Signal Buffer) ซึ่งช่วยให้คอมพิวเตอร์สามารถติดต่อส่งผ่านข้อมูลกับวงจรควบคุมความเร็วหรืออุปกรณ์ภายนอกได้ ผ่านแอดแดรสที่ต้องการได้โดยมีวิธีการกำหนดแอดแดรสดังนี้

การกำหนดแอดแดรสเพื่อติดต่อกับวงจรควบคุมความเร็ว(Speed Control)

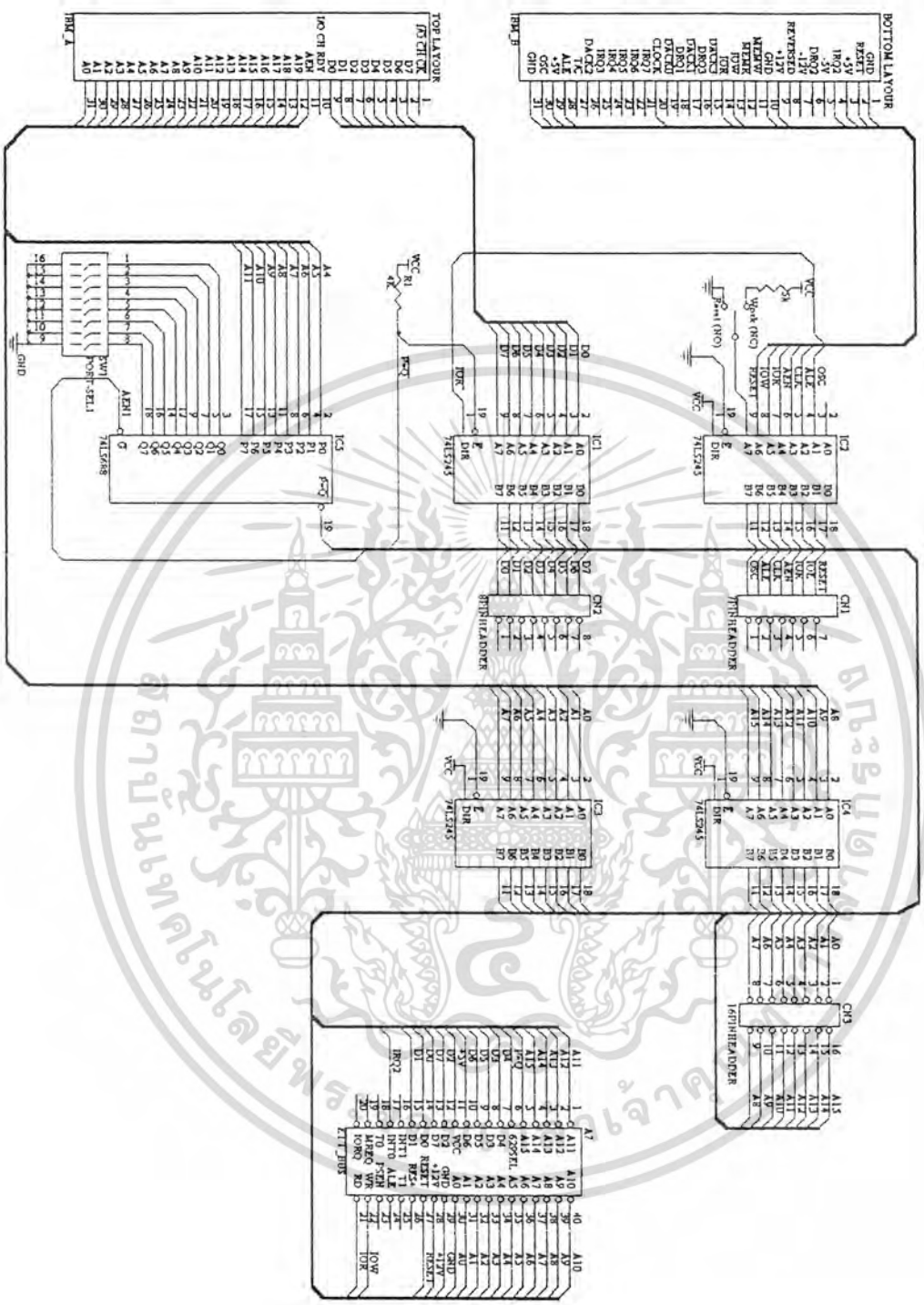
เนื่องจากเราใช้ ไอซีเบอร์ 74LS688 ในการดีโคดแอดแดรส (Decode Address) ดังนั้น เราจะสามารถเลือกใช้แอดแดรสเพื่อใช้ติดต่อกับวงจรควบคุมความเร็วได้จากตำแหน่งการบิด เปิดของดิปสวิทช์ (SW1,PORT SEL1) (ดูในวงจรรูปที่ 2.6) ตัวอย่างเช่น

ถ้าเราต้องการใช้แอดแดรส 300H (0011 0000 XXXX B) ในการติดต่อกับวงจรควบคุมความเร็ว หรือ A11= 0 , A10 = 0 , A9 = 1 , A8 = 1 , A7 = 0 , A6 = 0 , A5 = 0 , A4 = 0 ส่วน A3,A2,A1,A0 จะมีค่าเป็นอะไรก็ได้ ซึ่ง A1 - A3 จะถูกดีโคดด้วยไอซี 74LS138 อีกครั้งดังนั้นเราจะสามารถกำหนดตำแหน่งของดิปสวิทช์(Dip Switch) ได้ดังแสดงในรูปที่ 2.5



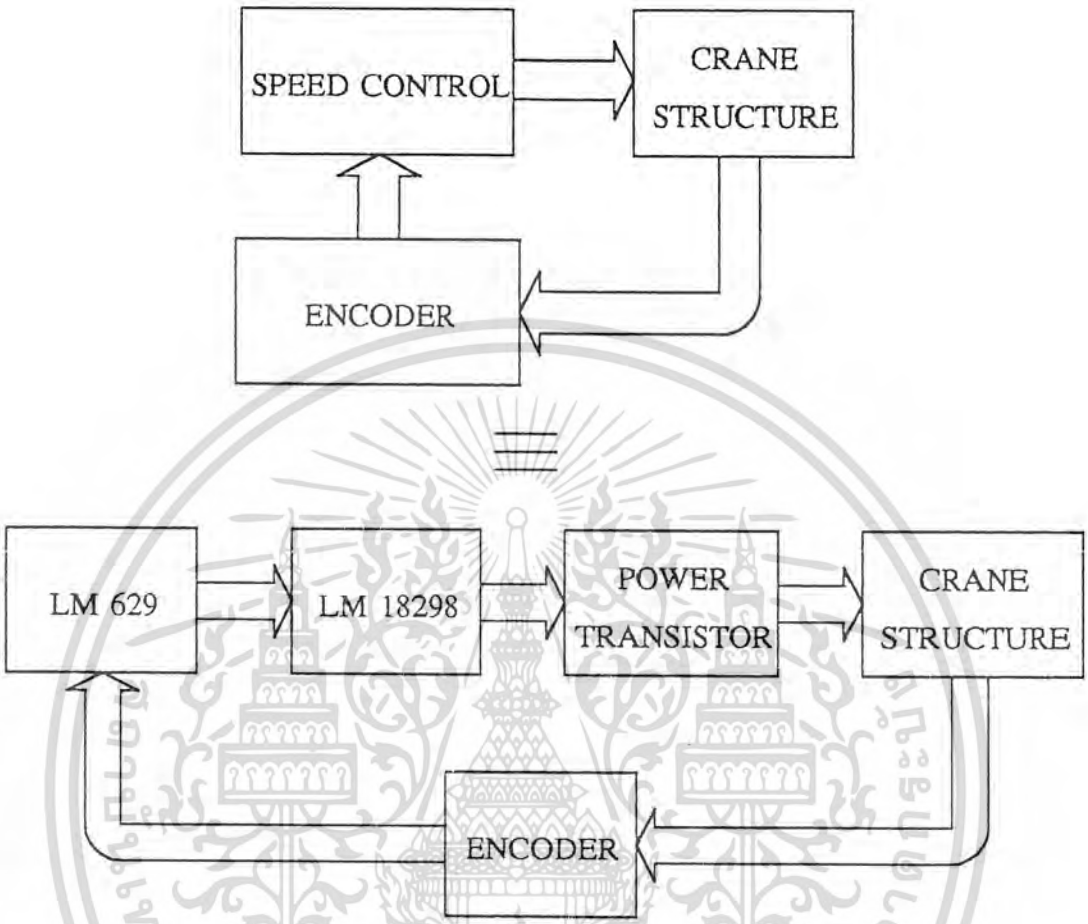
รูปที่ 2.5 รูปแสดงตัวอย่างการเลือกตำแหน่งของดิปสวิทช์

การกำหนดตำแหน่งดิปสวิทช์ดังกล่าวเราจะสามารถติดต่อกับ ชุดควบคุมความเร็วได้ 8 ชุด โดยการอ้างจาก A1 - A3 (ดูวงจรในรูปที่ 2.6) เพื่อติดต่อกับวงจรแต่ละชุด



รูปที่ 2.6 วงจรเชื่อมต่อระหว่างคอมพิวเตอร์และชุดควบคุมความเร็ว

2.3.1.3 ชุดควบคุมความเร็ว (Speed Control)



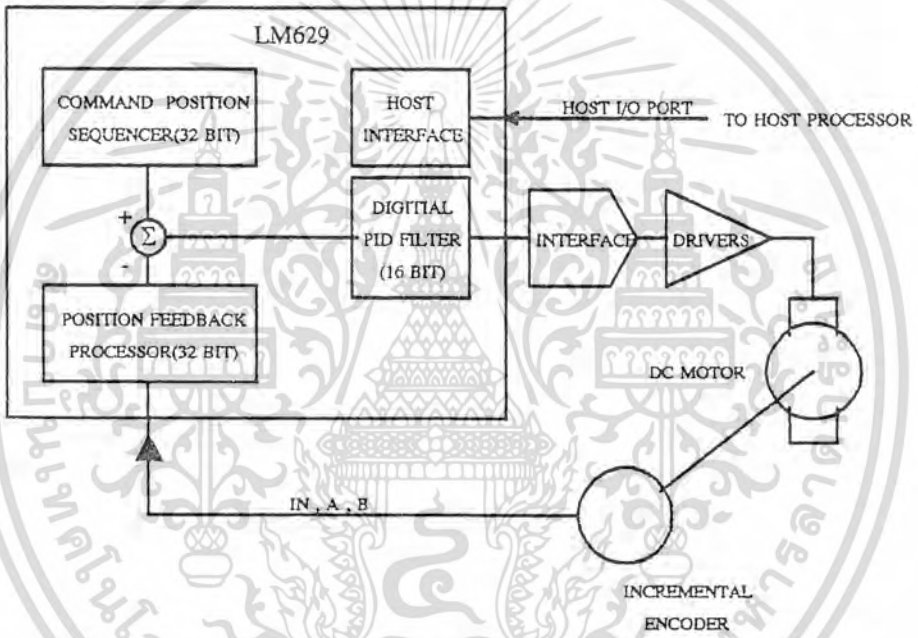
รูปที่ 2.7 บล็อคไดอะแกรมของชุดควบคุมความเร็ว

จากรูปที่ 2.7 จะเห็นว่าชุดควบคุมความเร็วนั้นประกอบไปด้วยส่วนต่างๆดังนี้

ก. ไอซี LM 629 เป็นชิพควบคุมการเคลื่อนที่ (Motion Control IC) ที่ต้องทำงานร่วมกับหน่วยประมวลผลหลัก (Host Processor)(ซึ่งในที่นี้ก็คือคอมพิวเตอร์) ที่ถูกออกแบบมาให้ใช้ได้กับทั้ง ดีซีเซอร์โวมอเตอร์ทั้งแบบมีแปรงถ่านและไม่มีแปรงถ่าน (Brush and Brushless DC Servo Motor) และระบบเซอร์โวแมคคานิสซึม (Servo Mechanism) อื่นๆ ที่มีการป้อนกลับสัญญาณตำแหน่งแบบ Quadrature Incremental เช่นจาก Quadratic Incremental Encoder โดยในการทำงานของ LM629 นั้นต้องใช้สัญญาณนาฬิกา ซึ่งในที่นี้เราใช้สัญญาณนาฬิกาความถี่ 6 MH_z ที่กำเนิดจากวงจรกำเนิดความถี่ ประกอบด้วย NAND GATE,R,C,CRYTAL 6 MH_z (ดูวงจรในรูป 2.9)

LM629 จะเชื่อมต่อกับคอมพิวเตอร์เพื่อรับคำสั่งความเร็วโดยผ่าน CON1 หรือ CON2 ซึ่งการ์ดอินเตอร์เฟสในหัวข้อที่ 2.3.1.2 ที่ทำหน้าที่เชื่อมต่อขาสัญญาณต่างๆจาก คอมพิวเตอร์กับ LM629 และ JUMPER1 จะใช้สำหรับเลือกว่าจะให้มีการ Interruptจาก LM629 หรือไม่ ส่วน

CON3 เป็นคอนเน็คเตอร์ที่ใช้ต่อกับ เอนโคดเดอร์โดยเอนโคดเดอร์ใช้ไฟเลี้ยงขนาด 12 V และให้เอาท์พุตปีนแบบ Open Corrector จึงต้องมีการ Pull up สัญญาณด้วยความต้านทานขนาด 5 กิโลโอมห์(ดูวงจรในรูป 2.9) สัญญาณที่ Pull up แล้วจะนำมาต่อกับขา IN,A,B ของ LM629 ตามลำดับ จากค่าสัญญาณป้อนกลับที่ได้ และค่าสัญญาณควบคุมจากคอมพิวเตอร์ LM 629 จะให้สัญญาณเอาท์พุตเป็น PWM (Pulse Width Modulate) Magnitude และ PWM Sign ขนาด 5 V TTL ซึ่งเป็นตัวกำหนดขนาดและทิศทางของแรงดันที่ป้อนให้มอเตอร์ซึ่งจะถูกนำไปขยายโดยวงจรภาคขับสัญญาณดังจะกล่าวในหัวข้อถัดไป (สามารถดูรายละเอียดของไอซีLM 629ได้จากภาคผนวก)



รูปที่2.8 รูปแสดงบล็อกไดอะแกรมของ LM629 กับตัวอย่างระบบที่ใช้งาน

<u>คุณสมบัติของ LM 629</u>

- มีรีจิสเตอร์ตำแหน่ง ความเร็ว และความเร่ง ขนาด 32 บิต
- เป็นฟิลเตอร์ดิจิตอลแบบ PID
- สามารถกำหนดเส้นทางของความเร็ว (Trajectory) ได้
- สามารถกำหนด ค่าคาบการชักตัวอย่างได้
- ให้เอาท์พุตเป็น PWM ซึ่งมีทั้งขนาดและทิศทาง
- ค่าความเร็ว ค่าตำแหน่งที่ต้องการ และค่าพารามิเตอร์ของฟิลเตอร์ สามารถเปลี่ยนแปลงได้ขณะที่ทำการควบคุมจริง
- สามารถทำงานได้ทั้งในโหมดการควบคุมตำแหน่งและโหมดการควบคุมความเร็ว
- สามารถโปรแกรม การทำการขัดจังหวะ(Interrupt) กับหน่วยประมวลผลหลักได้
- ติดต่อข้อมูลกับหน่วยประมวลผลหลักแบบขนาน ขนาด 8 บิต

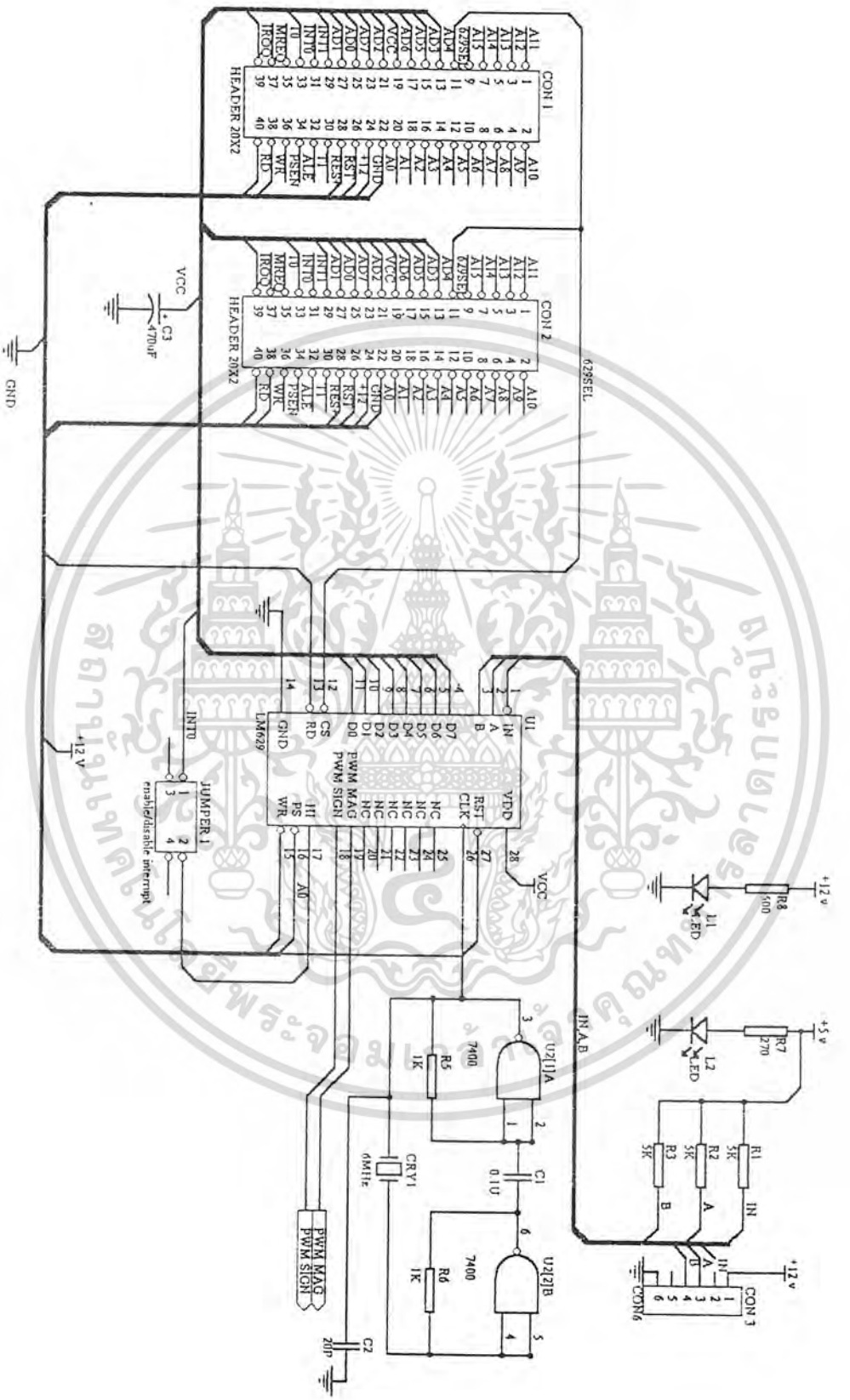
- มีหน่วยรับสัญญาณการป้อนกลับจาก เอ็นโคดเดอร์

การติดต่อกับ LM 629 การติดต่อกับ LM 629 หนึ่งตัวนั้นมีอยู่ 2 แบบ(ดูรายละเอียดในภาคผนวก) ดังนี้

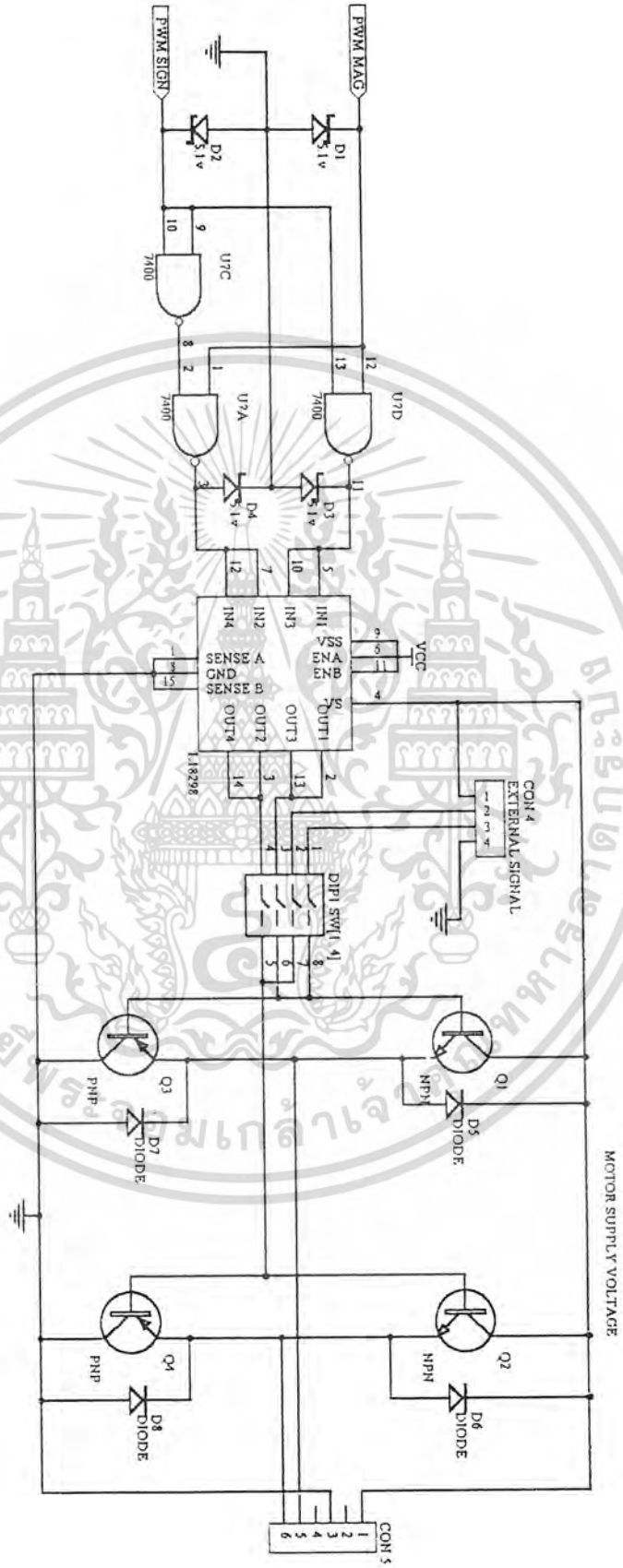
แบบที่1 เป็นการติดต่อเพื่อการส่งคำสั่งสั่งงาน LM 629 และเพื่อการอ่านค่าแฟล็กแสดงสถานะของ LM 629 ในการติดต่อแบบนี้เราจะต้องให้สัญญาณที่ขาที่ 12 (Chip Select) ของ LM 629 เป็น 0 และ ที่ขาที่ 16 (Port Select) ของ LM 629 เป็น 0

แบบที่2 เป็นการติดต่อเพื่อการอ่านข้อมูลและการเขียนข้อมูล ในการติดต่อแบบนี้เราจะต้องให้สัญญาณที่ขาที่ 12 (Chip Select) ของ LM 629 เป็น 0 และ ที่ขาที่ 16 (Port Select) ของ LM 629 เป็น 1

เนื่องจากการติดต่อกับ LM 629 มีการติดต่อ 2 แบบเราจึงต้องใช้แอดแดรส 2 แอดแดรสในการติดต่อกับ LM 629 1 ตัว (ดูวงจรในรูปที่ 2.9ก และรูปที่ 2.9ข)

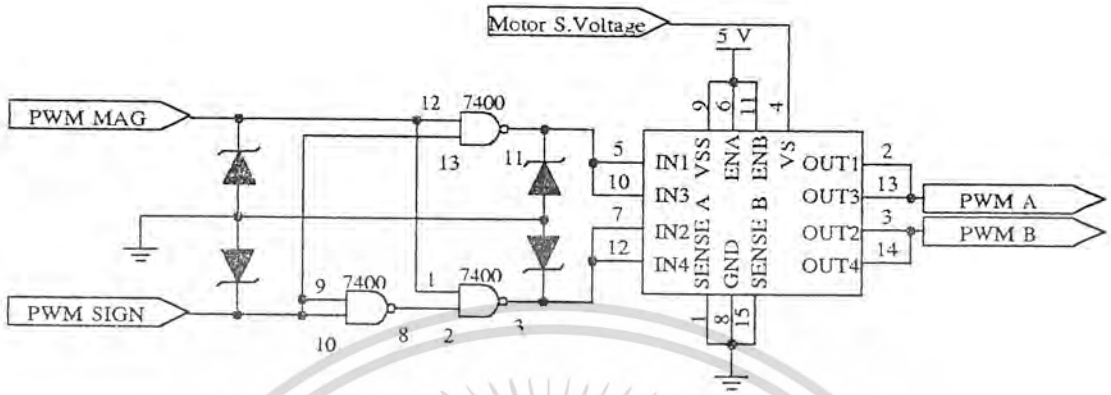


รูปที่ 2.9ก รูปแสดงวงจรของชุดควบคุมการเคลื่อนที่ของเครน



รูปที่ 2.9ข รูปแสดงวงจรของชุดควบคุมการเคลื่อนที่ของเครน(ต่อ)

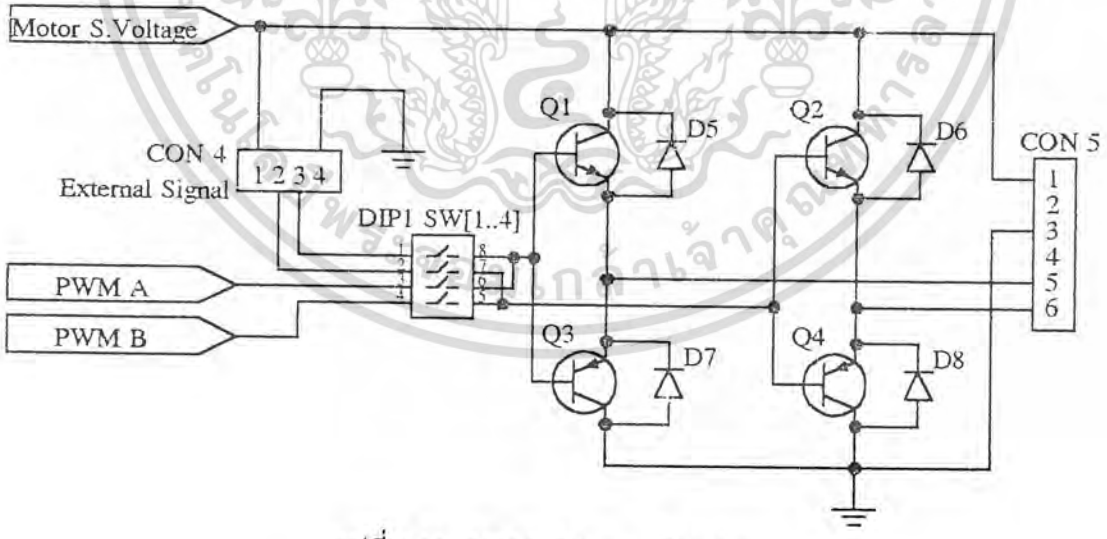
ข. LM18298



รูปที่ 2.10 รูปแสดงวงจร LM18298 ภาคขับPower Transistor

LM18298 ใช้เป็นภาคขับ Power Transistor โดยรับอินพุตเป็นสัญญาณ PWM Magnitude และ PWM Sign 5V TTL จาก LM 629 มาถอดรหัสและขยายสัญญาณให้เอาท์พุต เป็นสัญญาณ PWM ซึ่งจะมีค่าศักย์ดาไฟฟ้าเท่ากับค่าศักย์ดาที่จ่ายให้มอเตอร์ (Motor Supply Voltage,Motor S.Voltage) ที่จ่ายให้ LM18298 เพื่อนำไปขับ Power Transistor ต่อไป

ก. Power Transistor



รูปที่ 2.11 วงจรภาคPower Transistor

ประกอบด้วย Power Transistor 4 ตัวต่อเป็น Single Phase Full Bridge Drivers สามารถใช้ได้กับมอเตอร์ ไม่เกิน 140 V ใช้กระแสขับไม่เกิน 15 A โดยถ้า PWM Sign เป็น '0'

ด้วย Voltage เฉลี่ยแปรผันกับค่า Duty Cycle ของ PWM Magnitude ถ้าสัญญาณ PWM Sign เป็น '1' มอเตอร์ก็จะหมุนในทิศทางตรงกันข้าม และมีขนาดแปรตาม Duty Cycle ของ PWM Magnitude ดังอธิบายไว้ข้างต้น และในวงจรยังมีคอนเน็คเตอร์ (Connector) CON4 เป็นคอนเน็คเตอร์ที่ใช้รับสัญญาณควบคุมจากภายนอกโดยจะสามารถเลือกได้ว่าจะใช้สัญญาณจาก LM629 หรือ เอาท์พุทจากภายนอกมาขับมอเตอร์ ได้โดยปรับที่ DIP SW[1..4] (ดูวงจรในรูป2.9) และ CON5 เป็นคอนเน็คเตอร์ที่ใช้ต่อกับ มอเตอร์(ขาที่ 5 และ 6) และต่อกับแหล่งจ่ายไฟเลี้ยง Driver (ขาที่ 1 และ 3) โดยต้องไม่เกิน 46 V (โดยปกติใช้ 40 V) เพื่อไม่ให้ LM18298 เสียหาย

2.3.2 การเลือกใช้เซ็นเซอร์และออกแบบวงจรเซ็นเซอร์

เซ็นเซอร์เป็นส่วนประกอบที่สำคัญในระบบคอนโทรลที่มีการป้อนกลับ เพราะสัญญาณที่ป้อนกลับผ่านมาทางเซ็นเซอร์นั้น เราสามารถนำมาวิเคราะห์หาคุณสมบัติของระบบหรือสามารถบอกสถานะของระบบขณะนั้นได้ว่าเป็นอย่างไร ซึ่งจะได้นำมาหาสัญญาณควบคุมที่จะส่งไปควบคุมระบบ ดังนั้นถ้าสัญญาณป้อนกลับที่ได้ถูกต้องมากและมีสัญญาณรบกวนน้อย ก็จะทำให้เราได้ค่าสัญญาณควบคุมที่สามารถควบคุมระบบให้เกิดประสิทธิภาพสูงสุดได้

ในแบบจำลองเครนของเราใช้เซ็นเซอร์ 2 ตัว คือ

ก. โพเทนทิโอมิเตอร์ (Potentiometer)

ใช้เพื่อวัดค่ามุมการแกว่งของภาระ เราเลือกใช้โพเทนทิโอมิเตอร์แบบฟิล์มพลาสติก(ขนาดสูงสุด 2 กิโลโอมห์) เนื่องจากให้แรงดันเอาท์พุตที่เรียบ มีสแตติกน้อยส์ต่ำ ให้ความละเอียดสูง และค่าความต้านทานที่เกิดขึ้นจะมีความสัมพันธ์กับระยะทางเชิงมุมที่เกิดขึ้นเป็นเชิงเส้น

เราได้ทำการติดโพเทนทิโอมิเตอร์ไว้ที่ด้านล่างของตัวเครนและยึดแกนการหมุนของโพเทนทิโอมิเตอร์กับแกนของการแกว่งของภาระของเครน ทำให้เมื่อมีการแกว่งของภาระก็มีผลทำให้ค่าความต้านทานของโพเทนทิโอมิเตอร์เปลี่ยนไปด้วย ซึ่งค่าความต้านทานของโพเทนทิโอมิเตอร์จะมีความสัมพันธ์กับค่ามุมการแกว่งเป็นเชิงเส้น

<u>วงจรตรวจวัดค่ามุมโดยใช้ โพเทนทิโอมิเตอร์</u>

เนื่องจากค่าความต้านทานของโพเทนทิโอมิเตอร์จะมีความสัมพันธ์กับค่ามุมการแกว่งเป็นเชิงเส้น ดังนั้นเราจึงออกแบบวงจรที่ใช้ตรวจวัดค่ามุม โดยใช้โพเทนทิโอมิเตอร์ ซึ่งมีหน้าที่เปลี่ยนค่ามุมที่ได้มาเป็นสัญญาณโวลท์เตจ (แสดงวงจรในรูป 2.4) โดยใช้การจ่ายค่าแรงดันคงที่ให้กับโพเทนทิโอมิเตอร์ ค่าแรงดันตกคร่อมโพเทนทิโอมิเตอร์ที่ได้จะถูกนำมาเปรียบเทียบกับค่าแรงดันที่ได้จากค่าความต้านทานปรับค่าได้ R6 (ขนาดสูงสุด 10 กิโลโอมห์)(ดูรูป 2.4) ซึ่งใช้เป็นแรงดันอ้างอิง (Reference Voltage) และค่าแรงดันที่เปรียบเทียบมาได้จะถูกขยายสัญญาณโดย Instrument Amplifier ซึ่งมีอัตราขยายเป็นแบบ differential ที่สามารถปรับค่าอัตราขยายได้จากค่าความต้านทาน R_4 (ขนาดสูงสุด 5 กิโลโอมห์)และมีความสัมพันธ์ระหว่างค่าแรงดันเอาท์พุตและค่าแรงดัน

อินพุตดังนี้

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

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

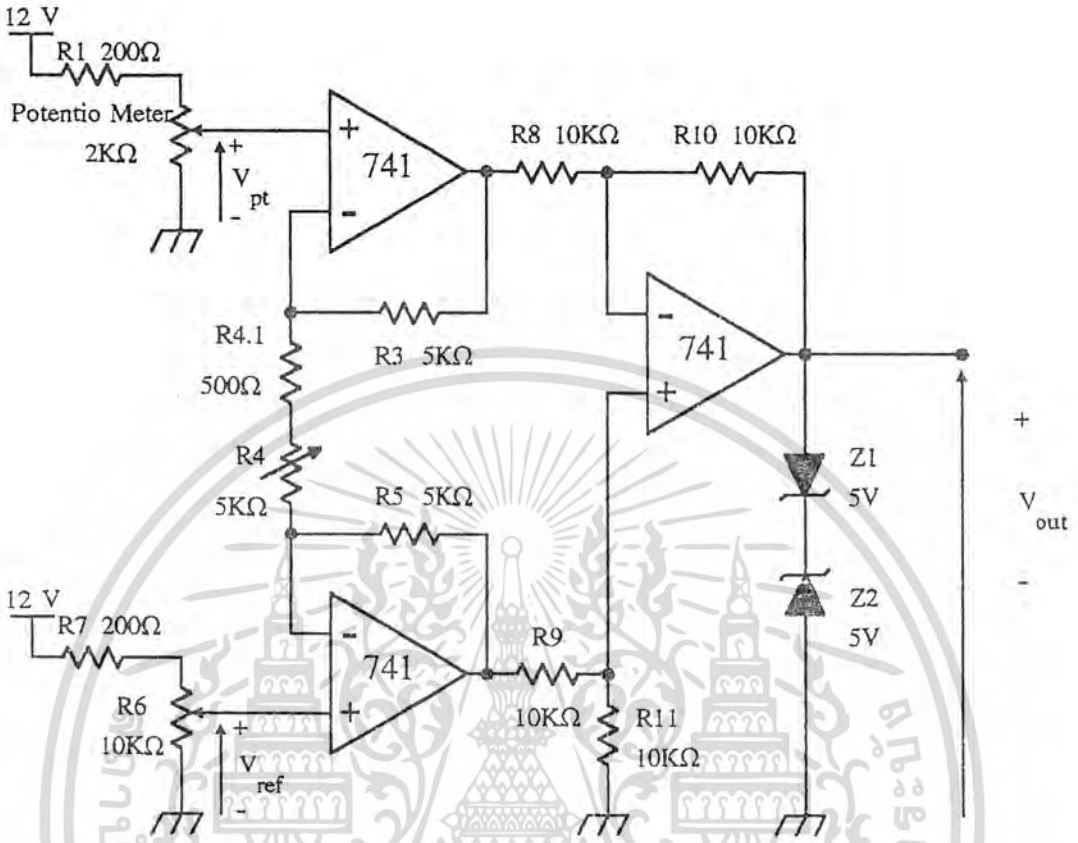
V_{out} คือค่าแรงดันเอาท์พุต และจะมีค่าอยู่ในช่วง -5 V ถึง +5 V

V_{pt} คือค่าแรงดันที่ได้จากการหมุนของโพเทนทิโอมิเตอร์

V_{ref} คือค่าแรงดันอ้างอิงที่ได้จากการปรับค่าความต้านทาน R6

จากความสัมพันธ์ในสมการจะเห็นว่า ค่าแรงดันเอาท์พุตนี้ (V_{out}) จะมีค่าสัมพันธ์กับแรงดันที่ได้จากค่าความต้านทานของโพเทนทิโอมิเตอร์ ดังนั้นค่า V_{out} ก็จะมีความสัมพันธ์กับค่ามุมของการแกว่งที่เกิดขึ้น เป็นเชิงเส้นเช่นกัน

หลังจากได้ค่ามุมป้อนกลับที่แปลงเป็นสัญญาณโวลท์เตจจากวงจรเซ็นเซอร์แล้วเราก็จะนำสัญญาณที่ได้มาผ่าน Low-Pass Filter ที่ออกแบบและทำขึ้นบนบอร์ด ACLD-780 Screw Terminal Panel เพื่อกรองสัญญาณรบกวนที่ความถี่สูงออก และก่อนจะทำการแปลงสัญญาณจาก อนาลอก เป็นดิจิตอลโดยผ่านการ์ด PCL-714 (ดูรายละเอียดในภาคผนวก) ที่ให้ความละเอียดของสัญญาณดิจิตอลถึง 14 บิต เพื่อทำให้สัญญาณที่ได้มาสามารถนำไปคำนวณในคอมพิวเตอร์ได้



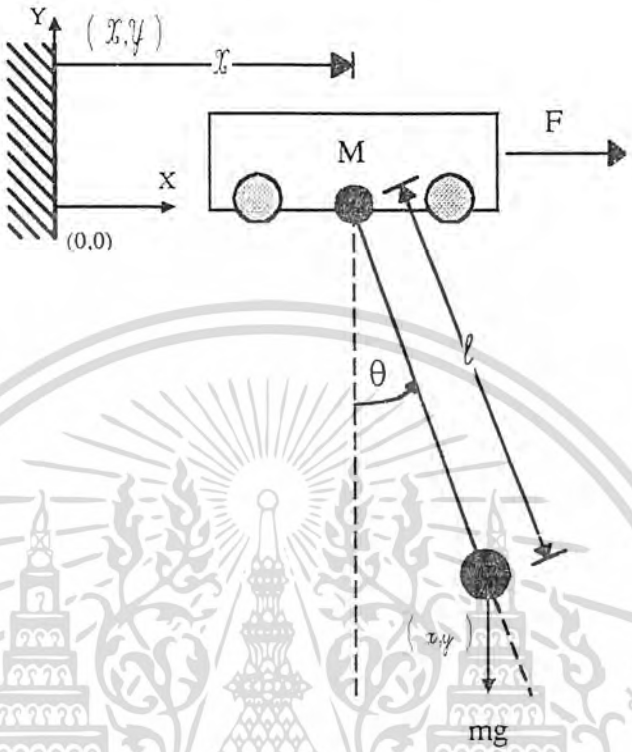
รูปที่ 2.12 รูปแสดงวงจรตรวจวัดค่ามุมโดยใช้โพเทนทิโอมิเตอร์

ข. เอนโคดเดอร์ (Encoder)

ใช้เพื่อวัดระยะทางของตัวเครน เราติดตั้งเอนโคดเดอร์ไว้ที่ด้านข้างของตัวเครน เมื่อเครนเคลื่อนที่ไปก็จะทำให้ลูกยางที่ยึดอยู่กับแกนของเอนโคดเดอร์เคลื่อนไปด้วย สัญญาณที่ได้จากการหมุนของเอนโคดเดอร์จะถูกป้อนเข้าไปที่ ไอซี LM629 (ดูไดอะแกรมในรูป2.8และวงจรในรูป2.9) และเราสามารถอ่านค่าระยะทางของเครนที่เคลื่อนที่ไปได้ โดยใช้คอมพิวเตอร์อ่านข้อมูลจาก รีจิสเตอร์ของไอซี LM629 (ดูรายละเอียดได้ในภาคผนวก)

2.4 โมเดลทางคณิตศาสตร์ของแบบจำลองเครน

2.4.1 การหาโมเดลทางคณิตศาสตร์ของแบบจำลองเครน



รูปที่ 2.13 รูปแสดงแบบจำลองของตัวเครน

จากรูปที่ 2.13 เมื่อ

M : มวลของตัวเครน (kg)

X : ระยะทางของเครนที่เคลื่อนที่ไปได้ตามแกนนอน เมื่อเทียบกับจุดอ้างอิง (m)

Y : ระยะทางของเครนที่เคลื่อนที่ไปได้ตามแกนตั้ง เมื่อเทียบกับจุดอ้างอิง (m)

m : มวลของภาระ (kg)

x : ระยะทางของภาระที่เคลื่อนที่ไปได้ตามแกนนอน เมื่อเทียบกับจุดอ้างอิง (m)

y : ระยะทางของภาระที่เคลื่อนที่ไปได้ตามตั้ง เมื่อเทียบกับจุดอ้างอิง (m)

l : ความยาวของเชือกที่แขวนภาระ (m)

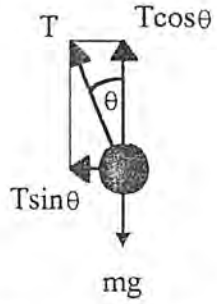
g : ความเร่งโน้มถ่วงของโลก (m/s^2)

F : แรงที่กระทำกับตัวเครน (N)

จะได้ความสัมพันธ์

$$x = X + l \sin\theta \qquad \text{----------(2.1)}$$

$$y = -l \cos\theta \qquad \text{----------(2.2)}$$



รูปที่ 2.14 Free Body Diagram ของภาระของเครน

จากรูปที่ 2.14 เมื่อ

T : แรงตึงในเส้นเชือก (N)

จะได้ความสัมพันธ์

$$m\ddot{x} = -T\sin\theta \qquad \text{----------(2.3)}$$

$$m\ddot{y} = T\cos\theta - mg \qquad \text{----------(2.4)}$$

และจากสมการ (2.3) และ (2.4) ได้

$$\ddot{x}\cos\theta + \ddot{y}\sin\theta = -g\sin\theta \qquad \text{----------(2.5)}$$

สมมติว่า l มีความยาวคงที่และหาอนุพันธ์อันดับสองของสมการ (2.1) และ (2.2) ได้

$$\ddot{x} = \ddot{\chi} - \dot{\theta}^2\sin\theta + l\,\ddot{\theta}\cos\theta \qquad \text{----------(2.6)}$$

$$\ddot{y} = l\,\dot{\theta}^2\cos\theta + l\,\ddot{\theta}\sin\theta \qquad \text{----------(2.7)}$$

แทนค่า และ จากสมการ (2.6) และ (2.7) ลงในสมการ (2.5)

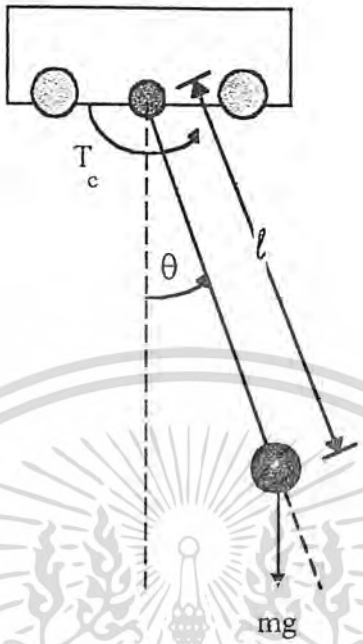
$$\ddot{\chi}\cos\theta + l\,\ddot{\theta} = -g\sin\theta$$

$$\ddot{\theta} = -\left(\frac{g}{l}\right)\sin\theta - \left(\frac{\ddot{\chi}}{l}\right)\cos\theta \qquad \text{----------(2.8)}$$

จากสมการ (2.8) สมมติให้มีการแกว่งของมุม θ มีค่าน้อยๆ ทำให้เราสามารถประมาณค่า $\sin\theta$ เป็น θ และประมาณ $\cos\theta$ เป็น 1 ทำให้ได้สมการที่มีความสัมพันธ์เป็นเชิงเส้น ดังนี้

$$\ddot{\theta} = -\left(\frac{g}{l}\right)\theta - \left(\frac{\ddot{\chi}}{l}\right) \qquad \text{----------(2.9)}$$

2.4.2 การหาความถี่ธรรมชาติของการแกว่งของภาระของเครน



รูปที่ 2.15 รูปแสดงการแกว่งของภาระของเครน

กำหนดให้

$$J = m\ell^2 \qquad \text{----------(2.10)}$$

จากรูป เมื่อ T_c : แอพพลายทอล์ค (N)

และ J : โมเมนต์ของแรงเฉื่อย(moment of inertia)ที่จุดหมุน ($kg\text{-}m^2$)

จะได้ผลรวมของโมเมนต์รอบจุดหมุนเป็น

$$T_c - mg\,\ell \sin\theta = J\ddot{\theta} \qquad \text{----------(2.11)}$$

แทนค่า J จากสมการ (2.10) ลงในสมการ (2.11) ได้

$$\ddot{\theta} + \frac{g}{\ell}\sin\theta = \frac{T_c}{m\ell^2} \qquad \text{----------(2.12)}$$

จากสมการ (2.12) สมมติให้มีการแกว่งของมุม θ มีค่าน้อยๆ ทำให้เราสามารถประมาณค่า sinθ เป็น θ และประมาณ cosθ เป็น 1 ทำให้ได้สมการที่มีความสัมพันธ์เป็นเชิงเส้น ดังนี้

$$\ddot{\theta} + \frac{g}{\ell}\theta = \frac{T_c}{m\ell^2} \qquad \text{----------(2.13)}$$

เราสามารถความถี่ธรรมชาติของการแกว่งของภาระของเครนได้จาก การสมมติให้แอพพลายทอล์ค T_c เป็นศูนย์

$$\omega_n = \sqrt{\frac{g}{\ell}} \qquad \text{----------(2.14)}$$

เมื่อ ω_n เป็นความถี่ธรรมชาติของการแกว่งชองภาระของเครน

บทที่ 3

การควบคุมมุมการแกว่งและตำแหน่งของเครน

ในการควบคุมการเคลื่อนที่ของเครนเราต้องการควบคุมตัวแปรหรือเอาท์พุตของระบบถึงสองค่าคือต้องการควบคุมตำแหน่งของเครนและต้องการควบคุมมุมการแกว่งของภาระให้มีค่าน้อยที่สุดเราจึงเลือกวิธีการควบคุมโดยการป้อนกลับสเตทแบบสเตทเรกกูเรเตอร์(State Regulator) ที่หาค่ากฎการควบคุมมาจากการนำสเตทที่ป้อนกลับ มาคูณด้วยค่าเกนป้อนกลับ K (Feedback Gain,K) ซึ่งทำการออกแบบค่าเกนป้อนกลับ K โดยวิธีการควบคุมออพติมัลแบบเรกกูเลเตอร์เชิงเส้นที่สถานะอยู่ตัว (Steady State Linear Quadratic Regulator) ที่มีดัชนีการทำงานแบบควอดเดรติกเป็น (Quadratic Performance Index) เพื่อให้ได้ผลตอบสนองดีที่สุดภายใต้ค่าดัชนีการทำงานที่ตั้งไว้ แต่เนื่องจากการควบคุมแบบการป้อนกลับสเตทนี้เราจำเป็นต้องเข้าถึงหรือรู้ค่าสเตททุกๆตัวที่เราสนใจที่ประกอบไปด้วย 1.มุมการแกว่งของภาระ(θ) 2.ความเร็วเชิงมุมของภาระ($\dot{\theta}$) 3. ตำแหน่งของเครน(x) 4.ความเร็วของเครน($\dot{x}$) 5.ความเร่งของเครน($\ddot{x}$) ซึ่งในทางปฏิบัติเราทำการวัดมาได้เพียงสองสเตทเท่านั้นคือ มุมการแกว่ง และ ตำแหน่งของเครนดังนั้นเราจึงจำเป็นต้องทำการประมาณค่าสเตทที่เหลือจากค่าสเตทที่เรารู้โดยเราเลือกมา สองวิธีคือ วิธีที่หนึ่งทำการประมาณค่าสเตทโดยอาศัยความความสัมพันธ์ทางฟิสิกส์ วิธีที่สองคือทำการออกแบบตัวสังเกตสเตท(State Observer)ที่ออกแบบโดยใช้คาลมานฟิลเตอร์(Kalman Filter)

3.1 การควบคุมออพติมัล (Optimal Control)

การออกแบบระบบควบคุมออพติมัล คือการหากฎการควบคุม(Control Law) เพื่อกำหนดการตัดสินใจสำหรับควบคุมในเวลาปัจจุบัน โดยมีเงื่อนไขบังคับบางประการเพื่อทำให้ระบบเบี่ยงเบนไปจากลักษณะที่ต้องการน้อยที่สุด ตัววัดการทำงานของระบบให้ได้ตามต้องการนี้เรียกว่าเกณฑ์การตัดสินใจหรือ ดัชนีการทำงาน (Performance Index, J) ซึ่งเป็นฟังก์ชันที่เราเลือกให้เป็นตัวชี้ว่าการทำงานจริงของระบบใกล้เคียงกับการทำงานที่ต้องการมากน้อยเพียงใด ดังนั้นปัญหาการควบคุมออพติมัลจึงอยู่ที่ การเลือกดัชนีการทำงาน และกฎควบคุมออพติมัล (Optimal Control Law, u_k) เพื่อออพติไมซ์(Optimize) ฟังก์ชั่นดัชนีการทำงาน หรือส่วนใหญ่จะเป็นการหาค่ากฎการควบคุมที่ทำให้ค่าดัชนีการทำงานมีค่าน้อยที่สุด(Minimization) นั่นเอง สำหรับการเลือกดัชนีการทำงานนั้นขึ้นกับแต่ละปัญหาเราไม่สามารถกำหนดเกณฑ์ในการเลือกดัชนีการทำงานให้ตายตัวได้

กำหนดระบบที่สามารถควบคุมได้โดยสมบูรณ์

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

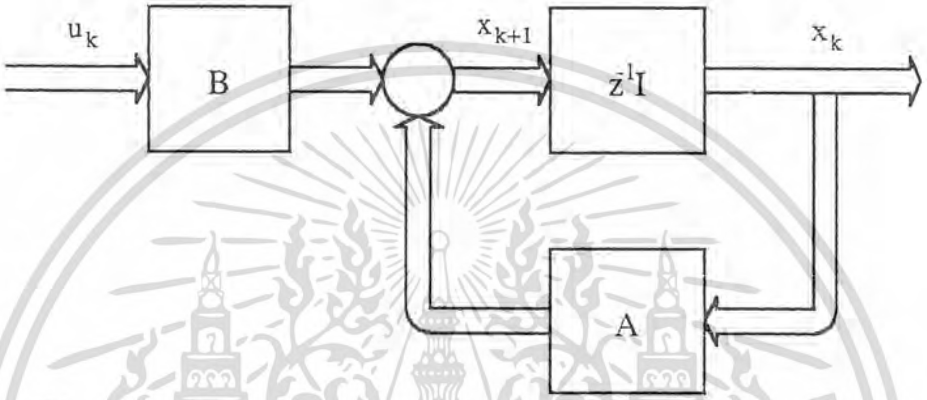
เมื่อ x_k เป็น n สเตทเวคเตอร์

u_k เป็น r เวคเตอร์ควบคุม

A เป็น (nxn) เมตริกซ์

B เป็น (nxr) เมตริกซ์

T หมายถึงทรานสโพส



รูปที่ 3.1 รูปแสดงบล็อกไดอะแกรมแบบของระบบ

ในที่นี้เราเลือกใช้ การควบคุมออพติมัลแบบ เรกกูเลเตอร์เชิงเส้นที่สถานะอยู่ตัว (Steady State Linear Quadratic 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 จะถูกเลือกเพื่อให้น้ำหนักความสำคัญของสเตทและสัญญาณอินพุต แต่ละตัวต่างๆกัน เมตริกซ์ Q จะถูกเลือกเพื่อจำกัดค่าผิดพลาดหรือขนาดของสเตทแต่ละตัว ยิ่งเราให้น้ำหนักความสัมคัญกับสเตทไหนมาก (เมื่อเปรียบเทียบกับที่ให้กับสเตทอื่น) ค่าผิดพลาดหรือขนาดของเสตทนั้นก็จะถูกจำกัดให้มีค่าน้อยลงเมื่อเทียบกับสเตทอื่น และเมตริกซ์ R จะถูกเลือกเพื่อจำกัดขนาดของสัญญาณควบคุมแต่ละตัว เช่นเดียวกันถ้าเราให้น้ำหนักความสัมคัญกับอินพุตตัวไหนมาก (เมื่อเทียบกับที่ให้กับอินพุตอื่น) ขนาดของอินพุตนั้นก็จะถูกจำกัดให้มีค่าน้อยลงเมื่อเทียบกับอินพุตตัวอื่น

การหาค่ากฎควบคุมออพติมัลนั้น สามารถหาได้หลายวิธีในที่นี้เราจะแสดงการหาค่ากฎควบคุมออพติมัล โดยการใช้ฟังก์ชันลีอาพูนอฟ (Liapunov Function)

3.1.1 การหากฎควบคุมออพติมัลสำหรับเรกกูเลเตอร์เชิงเส้นโดยใช้ฟังก์ชั่นลีอาพูนอฟ

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)) ต่อไปนี้น้อยที่สุด

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

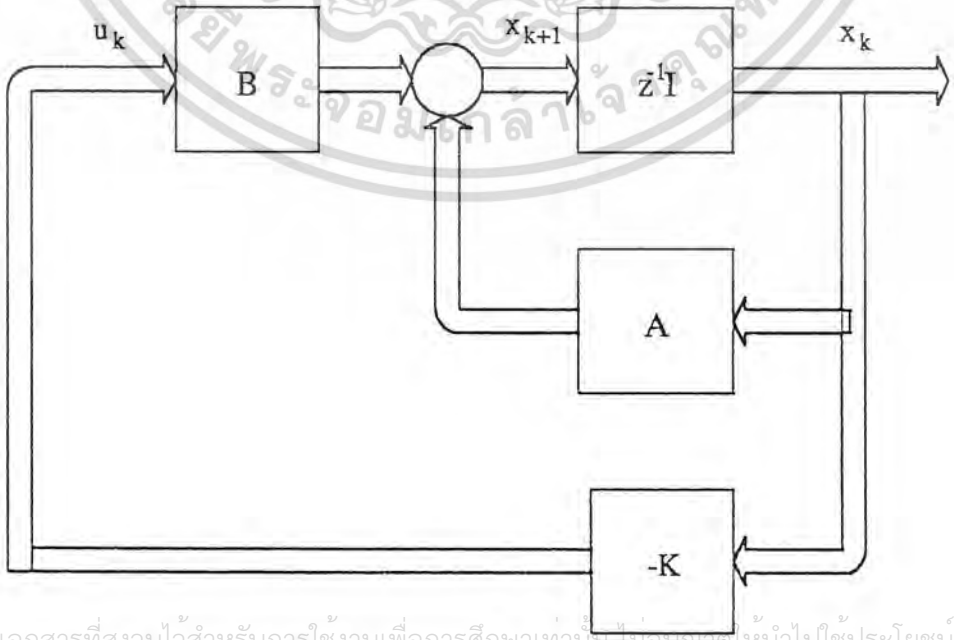
เนื่องจาก Q เป็นเมตริกซ์เฮอร์มิเชียน positive definite หรือ positive semidefinite และ R เป็นเมตริกซ์เฮอร์มิเชียน positive definite ดังนั้น ดัชนีการทำงาน J จึงเป็น positive definite

เราหาสมการของระบบวงปิดด้วยการป้อนกลับ $u_k = -Kx_k$ ได้

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

และ

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



รูปที่ 3.2 รูปแสดงบล็อกไดอะแกรมของระบบเมื่อป้อนกลับด้วย $u_k=-Kx_k$

ในที่นี้ให้ A-BK เป็นเมตริกซ์เสถียร ดังนั้นย่อมหาฟังก์ชั่นลีอาพูนอฟที่เป็น 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 $$\Delta V(x_k) = x_{k+1}^T P x_{k+1} - x_k^T P x_k$$

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 difinite 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

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

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

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

การทำให้ค่าดัชนีการทำงานควอดเดรติก น้อยที่สุดโดยคิดจาก เกนป้อนกลับ K ก็คือ การเลือกค่าเกน K ที่ทำให้ด้านซ้ายมือของสมการ(3.6) มีค่าน้อยที่สุด และ เนื่องจากเทอม $[(R+B^T PB)^{1/2} K-(R+B^T PB)^{-1/2} B^T PA]^T [(R+B^T PB)^{1/2} K-(R+B^T PB)^{-1/2} B^T PA]$ 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 ที่ทำให้ ดัชนีการทำงานควอดเดรติกมีค่าน้อยที่สุด

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

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

จากสมการ(3.7)และสมการ(3.8)เช่นเดียวกัน จะเห็นได้ว่าค่าของเมตริกซ์ P และของ

เมตริกซ์ K จะขึ้นกับค่าของเมตริกซ์ Q และเมตริกซ์ R ด้วย ดังนั้นถ้าสา มารถหนดเมตริกซ์ Q และเมตริกซ์ R ได้เหมาะสมกับระบบแล้ว เราก็จะได้ค่า เกนป้อนกลับ K ที่ทำให้ระบบมีผลตอบสนองดีที่สุด

จากสมการ(3.7)และสมการ(3.8)ที่ได้แสดงให้เห็นว่า ถ้าระบบ สามารถควบคุมได้โดยสมบูรณ์ และเราสามารถวัดค่าจากทุกสเตทได้ เมื่อเราหาโมเดลของระบบได้ และกำหนดเมตริกส์ Q และ เมตริกซ์ R เพื่อให้น้ำหนักความสำคัญกับสเตทและอินพุตแต่ละตัวแล้ว เราจะสามารถแก้สมการ(3.8) หาค่าเมตริกซ์ P ได้ และนำค่าเมตริกซ์ P ไปหาค่า เกน K ป้อนกลับสำหรับระบบเพื่อนำไปหาค่ากฎควบคุม ณ เวลาต่างๆได้

3.2 การประมาณค่าสเตทโดยใช้ดิสครีตคาลมานฟิลเตอร์ (Discrete Kalman Filter)

ในระบบที่ใช้ Deterministic Estimate หรือ Observers นั้นจะสมมติให้ระบบมีสมการทางคณิตศาสาตร์ที่แน่นอนตายตัวซึ่งไม่มี noise เข้ามาปะปนในระบบแต่ตามปกติแล้วระบบส่วนใหญ่จะประกอบด้วย process noise, measurement noise หรือทั้งสองอย่างรบกวนระบบอยู่

ในหัวข้อนี้เราจะใช้ ความรู้ทางความน่าจะเป็น เพื่ออธิบาย สัญญานรบกวนในระบบ ซึ่งได้สรุปเกี่ยวกับ ทฤษฎีความน่าจะเป็น(Probability Therory) ไว้ในภาคผนวกที่ ก

โดยตัวประมาณค่าสเตท เรียกว่าคาลมานฟิลเตอร์(Kalman Filter) นั้นคิดขึ้นครั้งแรกโดย R. Kalman[1960b] สำหรับระบบ Discrete และโดย Kalman and Bucy[1961] สำหรับระบบ Continuous-Time Linear Discrete Stochastic(Random) Systems ซึ่งระบบที่ประกอบด้วย noise แสดงได้โดย

$$x_{k+1} = Ax_k + Bu_k + Gw_k \qquad \text{----------(3.9)}$$

$$y_k = Cx_k + v_k \qquad \text{----------(3.10)}$$

โดย State $x_k \in R^n$, Control Input $u_k \in R^m$, Measured Output $y_k \in R^p$ และ process noise w_k แสดงถึง สัญญาณรบกวน หรือ Modeling Inaccuracies โดยเราจะสมมติให้ w_k เป็น white noise process ที่มี mean เป็นศูนย์และมี covariance = Q_f ซึ่งสามารถเขียนได้เป็น $w_k \sim (0,Q_f)$ และให้ measurement noise v_k เกิดจาก ความไม่เที่ยงตรงของเซ็นเซอร์ และเป็น stationary white noise ที่มี mean เป็นศูนย์ และ มี covariance เป็น R_f สามารถเขียนเป็น $v_k \sim (0,R_f)$

เราสมมติให้w_k กับv_k ไม่มีความสัมพันธ์ต่อกันดังนั้น $E\{w_j v_k^T\}=0$ สำหรับทุกค่า ของ j และ k นั่นคือ process noise และ measurement noise เกิดจากสาเหตุที่ไม่ขึ้นต่อกัน และเรากำหนดให้ $Q_f \geq 0$ และ $R_f > 0$ โดยที่ Q_f และ R_f เป็น Symmetric Matrix

เราเรียกสมการ (3.9) และ (3.10) ว่า Linear Stochastic System โดยสมมติให้ x_0 , w_j และ v_k ไม่มีความสัมพันธ์กัน

จุดประสงค์ของเราก็คือออกแบบตัวประมาณค่าสเตท(State Estimator)โดยวิธีคาลมานฟิลเตอร์ที่ให้ค่าประมาณของสเตท x_k โดยพิจารณาจากสภาพทาง Dynamic ของระบบในสมการ(3.9) และค่าที่วัดได้จากเอาท์พุตของระบบ ตามสมการ (3.10) ซึ่งจะแตกต่างจากตัวสังเกตสเตท(State Observer) ตรงที่คาลมานฟิลเตอร์จะใช้ความรู้เกี่ยวกับค่าทางสถิติของ noise เพื่อให้ค่าที่ประมาณได้ใกล้เคียงกับระบบในทางปฏิบัติมากขึ้น

ต่อไปเราจะมาพิจารณาลักษณะค่า mean ของสเตท และ การเปลี่ยนแปลงค่า covariance ของ process noise และ measurement noise

3.2.1 มีนและโควาเรียน (mean and covariance)

จากสมการที่ (3.9) เราสามารถพิจารณาค่า mean ของระบบได้ดังนี้

$$\overline{x}_{k+1} = \overline{Ax_k + Bu_k + Gw_k}$$

$$\overline{x}_{k+1} = A\overline{x}_k + B\overline{u}_k + G\overline{w}_k$$

เนื่องจาก $\overline{w}_k = 0$,เพราะเป็น white noise นั้นจึงได้

$$\overline{x}_{k+1} = A\overline{x}_k + Bu_k \qquad \text{----------(3.11)}$$

เขียน state covariance ได้ดังนี้ $P_{x_k} = E\{(x_k - \overline{x}_k)(x_k - \overline{x}_k)^T\}$ ซึ่งการเพิ่มขึ้นของค่า covariance สามารถเขียนได้ดังนี้

$$P_{x_{k+1}} = \overline{(x_{k+1} - \overline{x}_{k+1})\,(x_{k+1} - \overline{x}_{k+1})^T}$$

$$= \overline{[A(x_k - \overline{x}_k) + Gw_k]\,[A(x_k - \overline{x}_k) + Gw_k]^T}$$

$$= \overline{A(x_k - \overline{x}_k)\,(x_k - \overline{x}_k)^T A^T} + \overline{Gw_k (x_k - \overline{x}_k)^T A^T}$$

$$+ \overline{A(x_k - \overline{x}_k) w_k^T G^T} + \overline{Gw_k w_k^T G^T}$$

เพราะ x_k มีค่าขึ้นอยู่กับ x_0 , w_j , $j < k$, x_0 และ w_j ไม่สัมพันธ์กันดังนั้น

$$P_{x_{k+1}} = AP_{x_k}A^T + GQ_fG^T \qquad \text{----------(3.12)}$$

คอนที่เราจะมาพิจารณาค่า mean และ covariance ของ เอาท์พุต y_k จากสมการที่ (3.10) เนื่องจาก v_k เป็น white noise ดังนั้น $\overline{y}_k$ จะมีค่า

$$\overline{y}_k = C\overline{x}_k \qquad \text{----------(3.13)}$$

cross-covariance ระหว่าง สเตท และ เอาท์พุต คือ

$$P_{x_k y_k} = \overline{(x_k - \overline{x}_k)(y_k - \overline{y}_k)^T}$$

$$= \overline{(x_k - \overline{x}_k)[C(x_k - \overline{x}_k) + v_k]^T} \qquad \text{----------(3.14)}$$

$$= P_{x_k}C^T$$

สำหรับ covariance ของเอาท์พุต y_k เราสามารถหาได้โดย

$$P_{y_k} = \overline{(y_k - \bar{y}_k)(y_k - \bar{y}_k)^T}$$

$$= \overline{[C(x_k - \bar{x}_k) + v_k][C(x_k - \bar{x}_k) + v_k]^T}$$

or ----------(3.15)

$$P_{y_k} = CP_{x_k}C^T + R$$

สิ่งที่เราได้พิจารณามาแล้วนั้นแสดงได้ว่า เราสามารถคำนวณค่า mean และ covariance ของ สเตท และ เอาท์พุต ที่เวลา k+1 จากค่าของมันเองที่เวลา k โดยไม่ต้องนำค่าที่วัดได้ใดๆมาพิจารณา แต่ค่า $P_{k+1} > P_k$

3.2.2 การทำงานของคาลมานฟิลเตอร์

จุดมุ่งหมายของเราคือต้องการหาค่าประมาณของสเตท โดยใช้ข้อมูลต่างๆที่เรามีเพื่อประมาณค่าสเตทให้ใกล้เคียงกับค่า mean ที่สุด โดยค่า covariance ที่น้อยลงบ่งบอกถึงการที่สเตท x มีค่าใกล้เคียงค่า mean มากขึ้น ซึ่งตามปกตินั้น P_{k+1} จะมากกว่า P_k (จากสมการ(3.12)) เพราะ process noise ในเทอมของ w_k นั้นได้ถูกบวกเพิ่มเข้าไปในการชักตัวอย่าง(sampling) ต่อไปอย่างต่อเนื่องตลอดเวลา แต่เราสามารถใช้ค่าที่วัดได้ มาเพิ่มความถูกต้องในการประมาณค่าสเตทได้

สมมติให้ค่าสเตท ที่ประมาณได้ที่เวลา k ก่อนที่จะวัดค่า y_k แสดงด้วย $\hat{x}_{\bar{k}}$ เรียกว่า priori estimate

สมมติให้ค่าสเตท ที่ประมาณได้ที่เวลา k หลังจากทราบค่าของ y_k จากการวัดแล้ว แสดงด้วย $\hat{x}_k$ เรียกว่า posteriori estimate

ดังนั้น ดิสครีตคาลมานฟิลเตอร์ จะประกอบด้วย 2 ขั้นตอน ในแต่ละคาบเวลา k ขั้นตอนแรกคือ Time Update โดย $\hat{x}_{k-1}$ Update เป็น $\hat{x}_k$ ดังสมการ

$$\hat{x}_{\bar{k}} = A\hat{x}_{k-1} + Bu_{k-1}$$ ----------(3.16)

ขั้นตอนที่สองคือ Measurement Update โดยนำค่า เอาท์พุต y_k ที่วัดได้ ที่เวลา k มาคำนวณค่า $\hat{x}_k$ พิจารณาได้จากรูปที่ 3.3

ดังนั้น priori estimate ที่เวลา k หาได้จาก posteriori estimate ที่เวลา k-1 ค่า สเตท ที่ประมาณได้ จะมีความใกล้เคียงกับสเตทจริงน้อยลง ดังนั้น priori error covariance ที่เวลา k ปกติจะมากกว่า posterior error covariance ที่เวลา k-1 ซึ่งแสดงไว้ตามรูปที่ 3.3

เพราะฉะนั้นสิ่งที่จะต้องหาต่อไปคือในขั้นตอน Measurement Update ของคาลมานฟิลเตอร์ นั้นที่เวลา k มีข้อมูล 2 ชิ้นที่จะนำมาประกอบกันเพื่อประมาณค่าสเตท คือ priori estimate จาก (3.16) และค่าเอาท์พุต ที่วัดได้ y_k ต่อไปเราจะมาหาวิธีรวมข้อมูล 2 ชิ้นนี้ เพื่อนำไปประมาณค่า สเตท x_k ที่ต้องการ

3.2.3 การประมาณค่า Linear Mean-Square ของ x_k จากการพิจารณาค่า y_k

ก่อนอื่นสมมติสมการเพื่อประมาณค่าสเตท จากค่าเอาท์พุตที่วัดได้ โดยสมมติให้มีความสัมพันธ์เป็นเชิงเส้น

$$\hat{x}_k = Fy_k + g \qquad \text{----------(3.17)}$$

ในการหาค่าที่ดีที่สุดของ F และ g จะสามารถกระทำได้โดย Minimize ค่า Mean-Square Error $J = E\{\tilde{x}_k^T \tilde{x}_k\}$ โดยที่ $\tilde{x}_k = x_k - \hat{x}_k$ ให้มีค่าน้อยที่สุด ซึ่งเราจะใช้ความสัมพันธ์ของ Trace เพื่อ Minimize ค่าดังกล่าว

$$\begin{aligned}
J &= \overline{\mathrm{tr}(x_k - \hat{x}_k)^T(x_k - \hat{x}_k)} = \overline{\mathrm{tr}(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T} \\
&= \overline{\mathrm{tr}(x_k - Fy_k - g)(x_k - Fy_k - g)^T} \\
&= \overline{\mathrm{tr}[(x_k - \hat{x}_{\bar{k}}) - (Fy_k + g - \hat{x}_{\bar{k}})][(x_k - \hat{x}_{\bar{k}}) - (Fy_k + g - \hat{x}_{\bar{k}})]^T} \\
&= \mathrm{tr}[(P_{\bar{k}} + F(P_{y_k} + \bar{y}_k\bar{y}_k^T)F^T + (g - \hat{x}_{\bar{k}})(g - \hat{x}_{\bar{k}})^T + 2F\bar{y}_k(g - \hat{x}_{\bar{k}})^T - 2FP_{y_k x_k}]
\end{aligned}$$

โดย $\bar{y}_{k\hat{x}_{\bar{k}}} = C\hat{x}_{\bar{k}}$ and $P_{y_k x_k} = E\{(y_k - \bar{y}_k)(x_k - \hat{x}_{\bar{k}})\}^T$

โดยที่ $(d/dF)\mathrm{tr}(FHF^T) = 2FH$ และ $(d/dF)\mathrm{tr}(DFH) = D^T H^T$ สำหรับทุกๆ เมตริกซ์ F,D,H ดังนั้นในการ Minimum J จะสามารถกระทำได้โดย

$$\frac{\partial J}{\partial g} = 2(g - \hat{x}_{\bar{k}}) + 2F\bar{y}_k = 0$$

และ

$$\frac{\partial J}{\partial g} = 2F(P_{y_k} + \bar{y}_k\bar{y}_k^T) - 2P_{x_k y_k} + 2(g - \hat{x}_{\bar{k}})\bar{y}_k^T = 0$$

แก้สมการแรกเพื่อหาค่า g โดยได้

$$g = \hat{x}_{\bar{k}} - F\bar{y}_k$$

แทนค่า g ที่ได้ในสมการที่สอง

$$FP_{y_k} - P_{x_k y_k} = 0$$

ดังนั้น

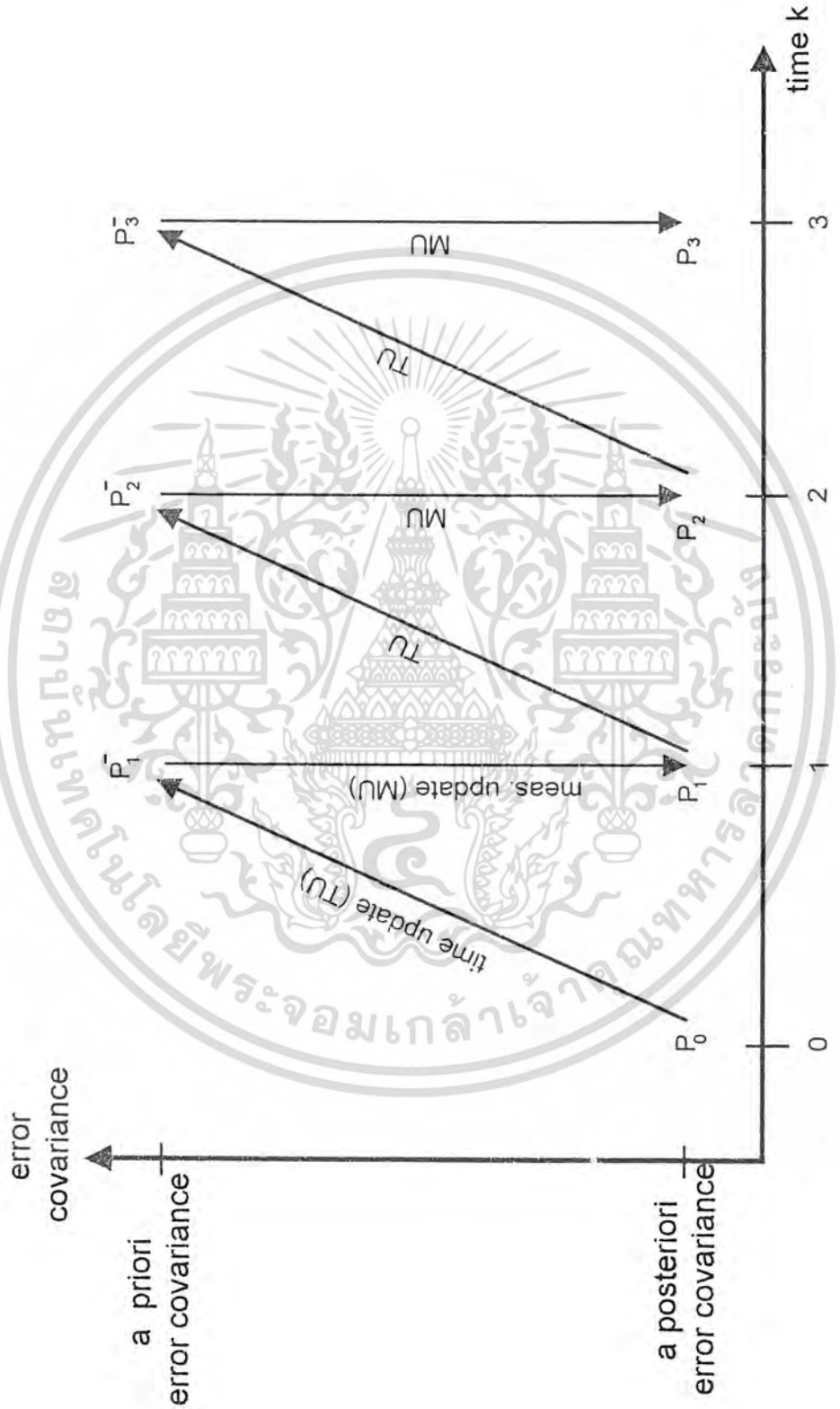
$$F = P_{x_k y_k} P_{y_k}^{-1} = 0$$

นำค่า g และ F ที่หาได้แทนใน สมการ(3.17) จะได้ การ Optimal Linear Mean-Square โดยใช้ค่าประมาณ สเตท $\hat{x}_{\bar{k}}$ ที่ได้จากคาบการชักตัวอย่างที่แล้วดังสมการ(3.16) และค่าเอาท์พุตที่วัดได้ y_k มาคำนวณซึ่งเป็นขั้นตอน Measurement Update ของ คาลมานฟิลเตอร์

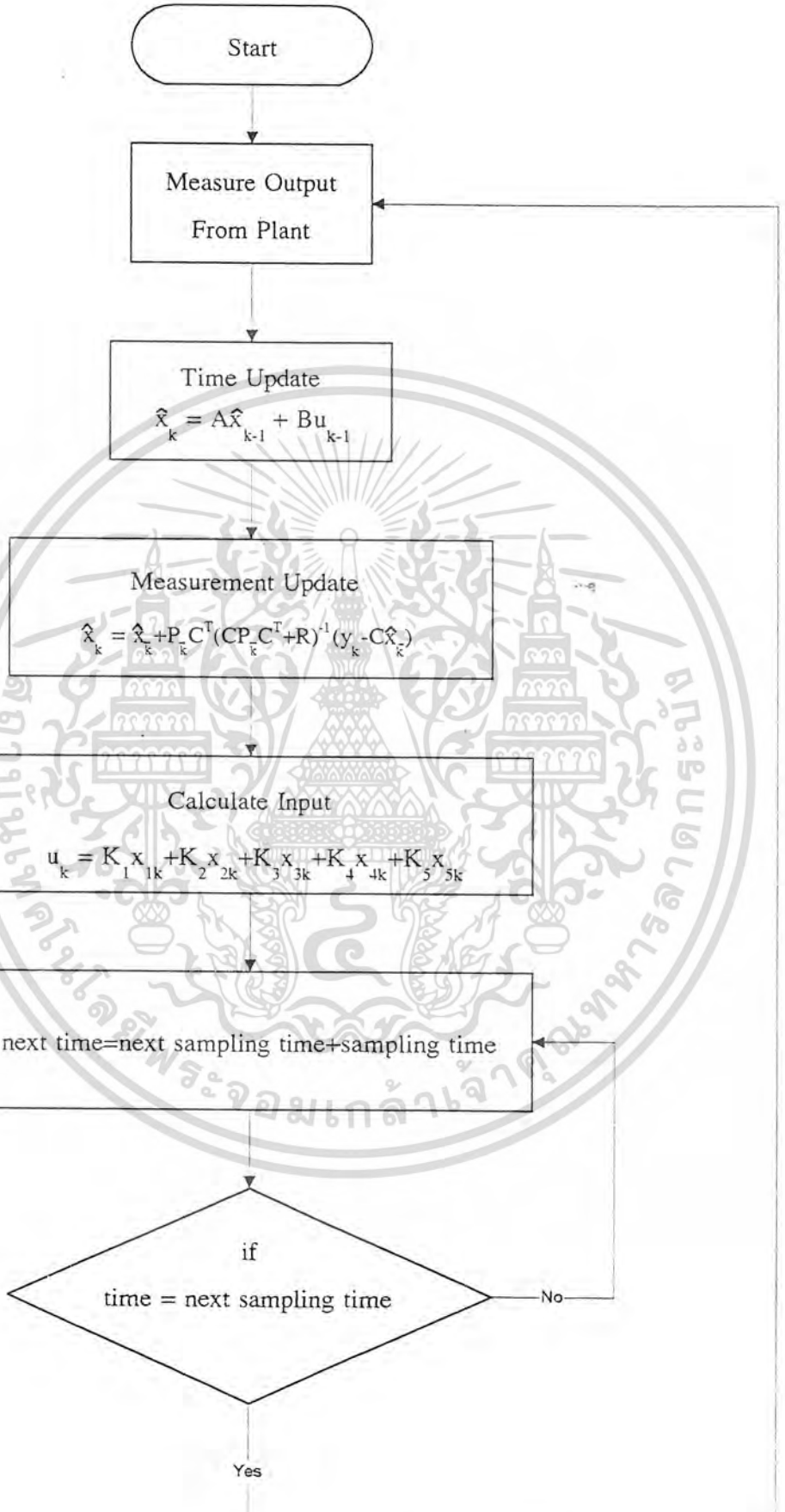
$$\hat{x}_k = \hat{x}_{\bar{k}} + P_{x_k y_k} P_{y_k}^{-1}(y_k - \bar{y}_k) \qquad \text{----------(3.18)}$$

$P_{x_k y_k}$, P_{y_k} และ $\bar{y}_k$ หาไว้แล้วจากสมการ (3.13,5),(3.14,6),(3.15,7)

$$\hat{x}_k = \hat{x}_{\bar{k}} + P_{\bar{k}}C^T(CP_{\bar{k}}C^T + R)^{-1}(y_k - C\hat{x}_{\bar{k}}) \qquad \text{----------(3.19)}$$



รูปที่ 3.3 รูปแสดงขั้นตอนแสดงการทำงานของคาลมานฟิลเตอร์



รูปที่ 3.4 กระบวนการทำงานของระบบที่ใช้คาลมานฟิลเตอร์

3.2.4 ความแปรปรวนของความผิดพลาดของสเตทที่ posteriori

ความแปรปรวนของความผิดพลาดของสเตทที่ประมาณค่าได้หลังจากนำค่าเอาท์พุตที่วัดได้มาประกอบในการคำนวณค่าสเตท (ที่ posteriori) ,P_k , นั้นถ้ายิ่งมีค่าน้อยจะหมายความว่าสเตทที่ประมาณได้จะถูกต้องมากขึ้น เพราะสเตทที่ประมาณได้จะเข้าใกล้ค่า mean มากขึ้น จากสมการที่ (3.18) เราสามารถนำมาช่วยในการเขียนสมการความผิดพลาดของการประมาณค่าสเตทได้ดังนี้

$$P_k = E\{\tilde{x}_k \tilde{x}_k^T\} = E\{(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T\}$$

$$= \left[(x_k - \hat{x}_{\bar{k}}) - P_{x_k y_k} P_{y_k}^{-1}(y_k - \bar{y}_k)\right]\left[(x_k - \hat{x}_{\bar{k}}) - P_{x_k y_k} P_{y_k}^{-1}(y_k - \bar{y}_k)\right]^T$$

$$= P_{\bar{k}} - P_{x_k y_k} P_{y_k}^{-1} P_{y_k x_k} - P_{x_k y_k} P_{y_k}^{-1} P_{y_k x_k} + P_{x_k y_k} P_{y_k}^{-1} P_{y_k} P_{y_k}^{-1} P_{y_k x_k}$$

หรือ

$$P_k = P_{\bar{k}} - P_{x_k y_k} P_{y_k}^{-1} P_{y_k x_k} \quad \text{----------(3.20)}$$

แทนค่า(3.14),(3.15)ลงในสมการที่ (3.20)จะได้

$$P_k = P_{\bar{k}} - P_{\bar{k}} C^T + (C P_{\bar{k}} C^T + R)^{-1} C P_{\bar{k}} \quad \text{----------(3.21)}$$

ซึ่งเราเรียกสมการข้างบนว่า Riccati Equation ของ P_k ซึ่งเมื่อเราแปลงโดยใช้สูตรอินเวอร์ทแบบ Lemma จะได้ Riccati Equation เป็นดังนี้

$$P_{\bar{k}} = [(P_{\bar{k}})^{-1} + (C^T R^{-1} C)^{-1}]^{-1} \quad \text{----------(3.22)}$$

จากสูตรผลรวมของความต้านทานที่ต่อขนานกันนั้นมีลักษณะเดียวกับสมการที่ (3.22) ซึ่งเราจะเห็นได้ว่า P_k จะน้อยกว่าทั้ง $P_{\bar{k}}$ และ $C^T R^{-1} C$ นั่นก็คือค่าความผิดพลาดของสเตทที่ประมาณได้หลังจากการนำเอาท์พุตมาคำนวณนั้นมีค่าน้อยกว่าค่าในช่วงที่ยังมิได้นำเอาท์พุตมาคำนวณ นำสมการ(3.22) แทนใน (3.12) ได้ความแปรปรวนของค่าความผิดพลาดดังนี้

$$P_{\bar{k+1}} = A[P_{\bar{k}} - P_{\bar{k}} C^T (C P_{\bar{k}} C^T + R)^{-1} C P_{\bar{k}}] A^T + G Q_f G^T \quad \text{----------(3.23)}$$

3.2.5 คาลมานฟิลเตอร์ที่สภาวะคงตัว (Suboptimum Steady-State Kalman Filter)

เมื่อระบบไม่แปรค่ากับเวลา (A,B,C คงที่) คาลมานฟิลเตอร์ก็ยังคงแปรค่าตามเวลาเนื่องจากเป็นฟังชั่นของ k ซึ่งทำให้เกิดปัญหาในการใช้งานจริงเนื่องจากจะต้องคำนวณค่าคาลมานเกน(Kalman Gain) $P_{\bar{k}} C^T (C P_{\bar{k}} C^T + R)^{-1}$ จากสมการที่ (3.19) ซึ่งถึงแม้จะสามารถคำนวณไว้ก่อนที่จะมีการควบคุมจริง ก็จะต้องใช้หน่วยความจำจำนวนมากในการเก็บค่าคาลมานเกน ที่คำนวณได้ ดังนั้นจะง่ายกว่าถ้าค่าคาลมานเกน เป็นค่าคงที่ ดังจะกล่าวต่อไปนี้

เมื่อพิจารณาที่สภาวะคงตัวค่าความแปรปรวนของสเตทจริงกับสเตทที่ประมาณค่าได้จะมีค่าคงที่

$$P \equiv P_{\bar{k+1}} = P_{\bar{k}} \quad \text{----------(3.24)}$$

ดังนั้นสมการ Riccati (3.23) สามารถเขียนได้ใหม่ที่สภาวะคงตัวดังนี้

$$P = A[P - PC^T(CPC^T(CPC^T + R)^{-1}CP]A^T + GQ_fG^T \text{----------(3.25)}$$

ลักษณะของคาลมานฟิลเตอร์ที่ได้นี้มีลักษณะคล้ายกับ Observer แต่แตกต่างกันที่เราใช้กระบวนการออกแบบโดยวิธีเฟ้นสุ่มโดยกำหนดค่าพารามิเตอร์ Q_f , R_f ซึ่งเป็น covariance ของ measurement noise และ process noise ตามลำดับเพื่อใช้ในการออกแบบฟิลเตอร์ ซึ่งฟิลเตอร์ที่ได้จากการออกแบบจะทำการประมาณค่าสเตทที่ใกล้เคียงที่สุด (Optimum Estimate) สำหรับ noise ที่มีความแปรปรวนดังที่ออกแบบไว้

ดังนั้นที่สภาวะคงตัวจะสามารถเขียนสมการที่ (3.19) ใหม่ได้ดังนี้

$$\hat{x}_k = \hat{x}_{\bar{k}} + PC^T(CPC^T + R)^{-1}(y_k - C\hat{x}_{\bar{k}})$$

โดยให้ $K = P_{\bar{k}}C^T(CP_{\bar{k}}C^T + R)^{-1}$

$$\hat{x}_k = \hat{x}_{\bar{k}} + K(y_k - C\hat{x}_{\bar{k}}) \text{----------(3.26)}$$

ซึ่งเมตริกคงที่ คาลมานเกน(Kalman Gain) สามารถหาได้โดยคำนวณไว้ก่อนโดยใช้โปรแกรม MATLAB แล้วนำไปเก็บไว้ในหน่วยความจำเพื่อนำมาใช้ในระหว่างทำการควบคุมจริง

ในการวิเคราะพฤติกรรมของคาลมานฟิลเตอร์ที่สภาวะคงตัวนั้น เราจะพิจารณาจากค่าความผิดพลาดที่สภาวะคงตัว(Steady-State Error) ของระบบ ซึ่งก็คือ $\tilde{x}_k = x_k - \hat{x}_k$ ขั้นแรกแทนสมการ (3.26) ลงในขั้นตอน Time Update , $\hat{x}_{\bar{k+1}} = A\hat{x}_k + Bu_k$, ซึ่งเราจะทำการประมาณค่าสเตทต่อไปจากสเตทที่แล้วได้ดังนี้

$$\hat{x}_{\bar{k+1}} = A\hat{x}_{\bar{k}} + Bu_k + AK(y_k - C\hat{x}_{\bar{k}}) \text{----------(3.27)}$$

และจาก (3.9),(3.27),(3.10) จะได้ค่าความผิดพลาดที่สภาวะคงตัวของระบบที่เวลา priori ดังนี้

$$\tilde{x}_{\bar{k+1}} = A(I-KC)\tilde{x}_{\bar{k}} + Gw_k - AKv_k \text{----------(3.28)}$$

จากสมการที่ (3.28) จะเห็นได้ว่าความผิดพลาดที่สภาวะคงตัวของระบบเกิดจากทั้ง process noise w_k และ measurement noise v_k ซึ่งถ้า

$$A_0 = A - AKC \equiv A - LC \text{----------(3.29)}$$

เสถียรแล้วก็จะทำให้ค่าผิดพลาดที่สภาวะคงตัวลู่เข้า โดยจะปรากฏเฉพาะเทอม w_k และ v_k ที่สภาวะคงตัว ดังนั้นขนาดของค่าผิดพลาดจะขึ้นอยู่กับทั้งความเสถียรของ A_0 ,ขนาดของ w_k และ v_k ดังนั้นยิ่ง A_0 เสถียรภาพมากเท่าใดค่าความผิดพลาดของการประมาณค่าสเตทก็จะยิ่งลู่เข้าเร็วขึ้นเท่านั้น

<u>3.3 การประมาณค่าสเตทโดยอาศัยความสัมพันธ์ทางฟิสิกส์</u>

การประมาณค่าสเตทโดยวิธีนี้ทำได้ง่ายๆดังนี้

1.การประมาณค่าความเร็วเชิงมุมของภาระของเครน

จากความสัมพันธ์ของความเร็วเชิงมุมของภาระ($\dot{\theta}$)กับค่ามุมการแกว่งของภาระ(θ)

$\dot{\theta}$ = อัตราการเปลี่ยนแปลงของมุมการแกว่งของภาระ(θ)ในหนึ่งหน่วยเวลา

ดังนั้นความเร็วเชิงมุมของภาระในช่วงคาบเวลาการชักตัวอย่างที่ k ($\dot{\theta}_k$)จะมีค่าเป็น

$$\dot{\theta}_k = (\theta_k - \theta_{k-1})/(\text{sampling time}) \qquad \text{----------(3.30)}$$

เมื่อ θ_k คือค่ามุมการแกว่งที่วัดได้ในการชักตัวอย่างที่ k

θ_{k-1} คือค่ามุมการแกว่งที่วัดได้ในการชักตัวอย่างที่ k-1

sampling time คือค่าคาบเวลาที่ใช้ในการชักตัวอย่าง

2.การประมาณค่าความเร็วของเครน

จากความสัมพันธ์ของความเร็วของเครน($\dot{x}$)กับตำแหน่งของเครน(x)

$\dot{x}$ = อัตราการเปลี่ยนแปลงของตำแหน่งของเครน(x)ในหนึ่งหน่วยเวลา

ดังนั้นความเร็วของเครนในช่วงคาบเวลาการชักตัวอย่างที่ k ($\dot{x}_k$)จะมีค่าเป็น

$$\dot{x}_k = (x_k - x_{k-1})/(\text{sampling time}) \qquad \text{----------(3.31)}$$

เมื่อ x_k คือค่าตำแหน่งของเครนที่วัดได้ในการชักตัวอย่างที่ k

x_{k-1} คือค่าตำแหน่งของเครนที่วัดได้ในการชักตัวอย่างที่ k-1

3.การประมาณค่าความเร่งของเครน

จากความสัมพันธ์ของความเร่งของเครน ($\ddot{x}$) กับความเร็วของเครน($\dot{x}$)

$\ddot{x}$ = อัตราการเปลี่ยนแปลงของความเร็วของเครน($\dot{x}$)ในหนึ่งหน่วยเวลา

ดังนั้นความเร่งของเครนในช่วงคาบเวลาการชักตัวอย่างที่ k ($\ddot{x}_k$)จะมีค่าเป็น

$$\ddot{x}_k = (\dot{x}_k - \dot{x}_{k-1})/(\text{sampling time}) \qquad \text{----------(3.32)}$$

เมื่อ $\dot{x}_k$ คือค่าความเร็วของเครนที่วัดได้ในการชักตัวอย่างที่ k

$\dot{x}_{k-1}$ คือค่าความเร็วของเครนที่วัดได้ในการชักตัวอย่างที่ k-1

บทที่ 4

การประมาณค่าโมเดลของระบบจากผลตอบสนองที่ได้การทดลอง

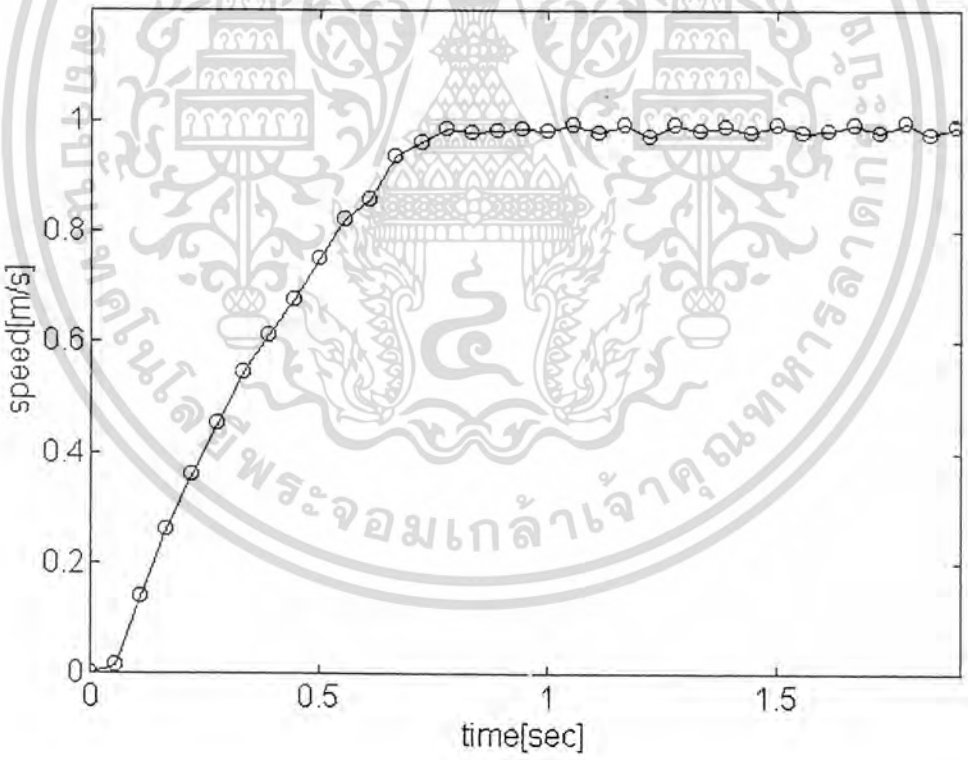
4.1 การหาโมเดลของมอเตอร์

จากการพิจารณา ผลตอบสนองของมอเตอร์ต่อ Step Input 40 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

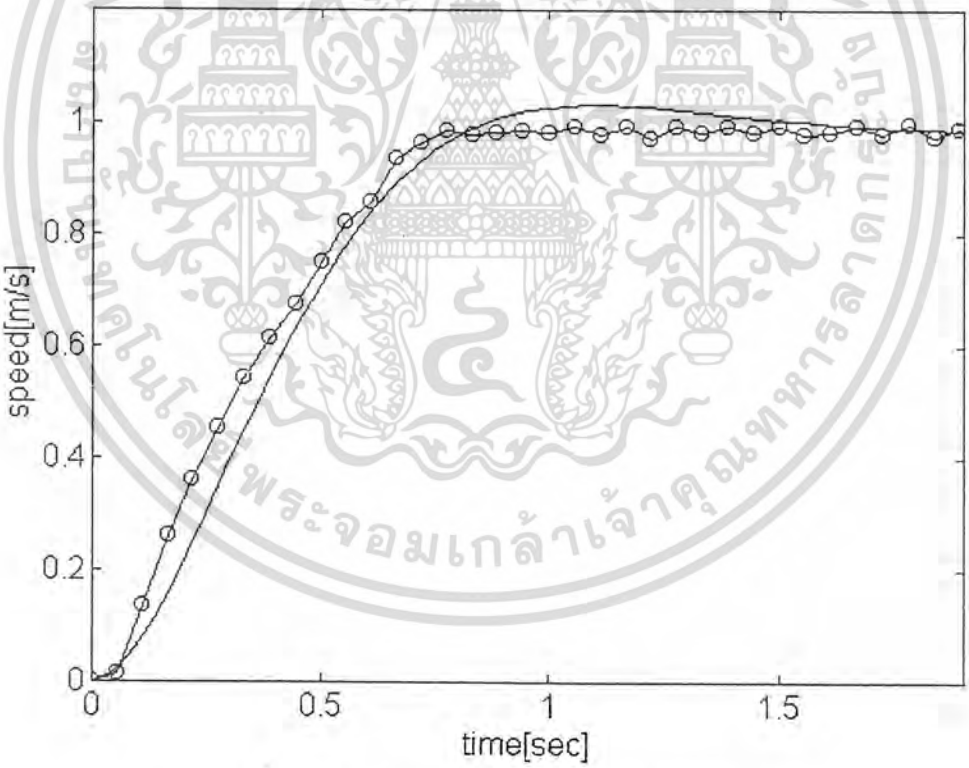
จากผลตอบสนองที่ได้จากการทดลอง ให้ Step Input 40 V เราสามารถประมาณค่าพารามิเตอร์ต่างๆของระบบได้โดยปรับค่า ζ และ ω_n เพื่อให้ได้ผลตอบสนองที่ใกล้เคียงกับผลตอบสนองที่ได้จากระบบจริงมากที่สุด โดยพิจารณาจากกราฟผลตอบสนองของระบบจริงกับผลตอบสนองที่ได้จากการ Simulate บน MATLAB ซึ่งได้ผลดังนี้

ζ=0.7 ω_n=3.98 K=0.985

ดังนั้นเราจะได้ Transfer Function ของมอเตอร์ดังนี้

$$\frac{C(s)}{R(s)} = \frac{15.6}{s^2 + 5.572s + 15.874} \qquad \text{----------(4.3)}$$

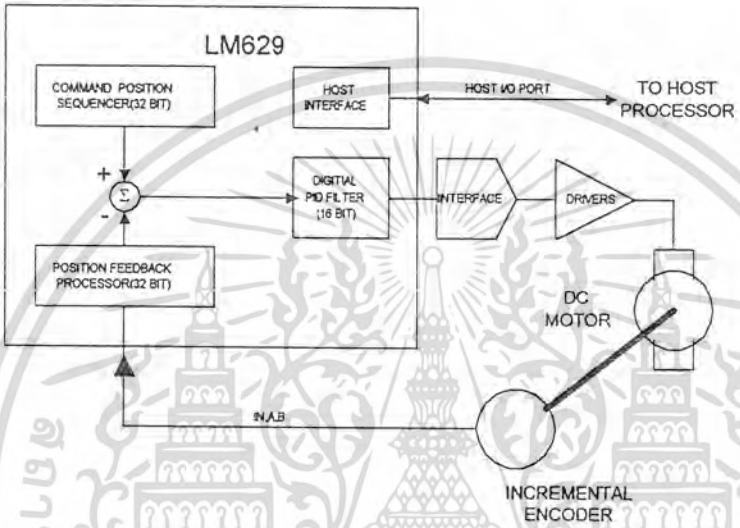
ซึ่งผลตอบสนองที่ได้จากการ Simulate เปรียบเทียบกับค่าที่วัดได้จริงดังรูปที่ 4.2



รูปที่ 4.2 ผลตอบสนองความเร็วต่อ Step Voltage
เปรียบเทียบกับ Model ที่ประมาณได้
โดย o-o ผลตอบสนองจริง , -- Model

4.2 การคำนวนค่า PID Controller

โดยเราจะใช้ IC LM629 เป็นตัวควบคุมความเร็วของมอเตอร์ ซึ่งภายในตัว LM629 นั้นจะทำงานเป็น PID Controller ดังรูปที่ 4.3 ดังนั้น เราจึงต้องคำนวนค่าพารามิเตอร์ K_i , K_p , K_d ที่เหมาะสมกับTransfer Function ของมอเตอร์แล้วนำค่าพารามีเตอร์ที่ได้โปรแกรมลงใน LM629 ในส่วน Digital PID Controller Filter ดังในรูปที่ (4.3) เพื่อให้ LM629 ทำหน้าที่เป็น PID Controller ในลูปความเร็วของรถเครน



รูปที่ 4.3 โครงสร้างภายในของ LM629 PID Controller

โดยเราจะทำการออกแบบค่า K_i , K_p , K_d ที่ต้องป้อนให้ LM629 โดยวิธีของ Ziegle-Nichols ซึ่งขั้นแรกนั้นจะสมมติให้ K_i=K_d=0 แล้วเพิ่ม K_p จนกระทั่งระบบเริ่มแกว่ง(Oscillate) ค่า Proportional Gain :K_p จะเป็น 0.6 เท่าของค่า Gain ณ ตำแหน่งนั้น และค่า K_i ,K_d สามารถคำนวณได้ดังนี้

$$K_p=0.6K_m \qquad K_d=\frac{K_p \pi}{4\omega_m} \qquad K_i=\frac{K_p \omega_m}{\pi} \quad \text{-------(4.4)}$$

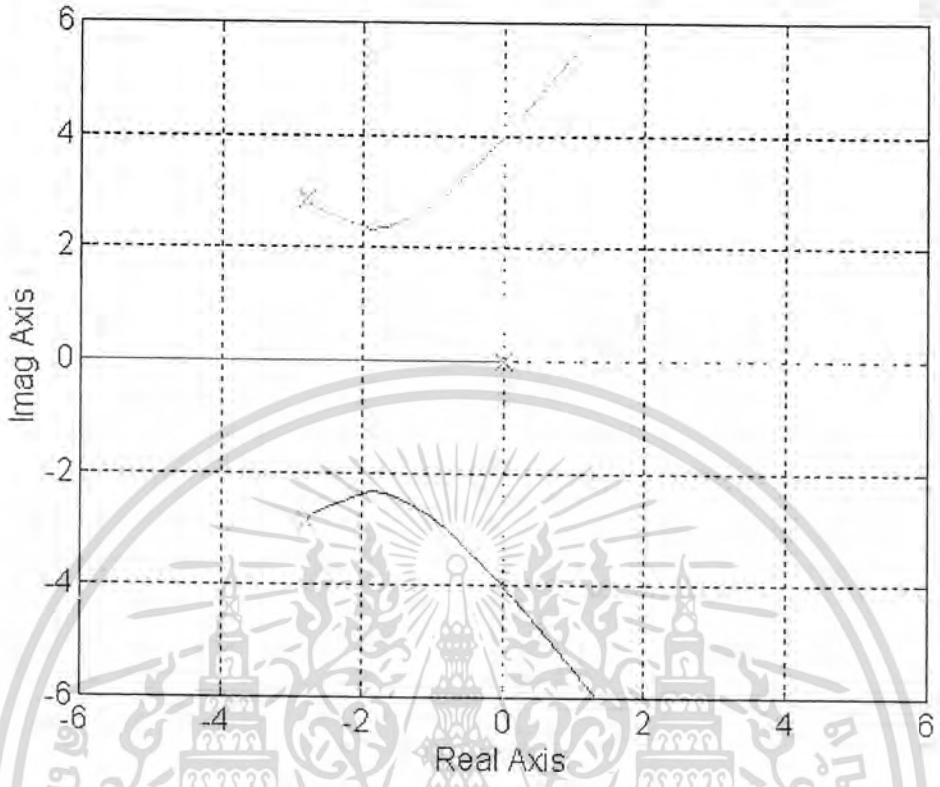
K_m = ค่า Gain ที่ระบบเริ่มแกว่ง

ω_m = ความถี่ที่ระบบเริ่มแกว่ง

จาก Root Locus ดังรูปที่ 4.4 จุดที่ทางเดินของรากตัดกับแกนจินตภาพคือจุดที่ระบบเริ่มแกว่ง ซึ่งมีค่า K_m = 5.8 และ ω_m = 4 rad/sec นำมาแทนในสมการที่ (4.4) ได้ค่าพารามิเตอร์ต่างๆ ดังนี้

K_p = 3.48 K_d = 0.68 K_i = 4.43

จากนั้นนำพารามิเตอร์ที่ได้โปรแกรมลงใน LM629 ซึ่งรายละเอียดได้กล่าวไว้ในภาคผนวก ค



รูปที่ 4.4 แสดงทางเดินของราก

4.3 การหาโมเดลของ LM629 รวมกับ มอเตอร์

ต่อจากนั้นนำค่าพารามิเตอร์ที่คำนวนได้โปรแกรมลงใน LM629 ซึ่งเป็น PID Controller ใน Speed Loop เมื่อเราทดลองป้อนคำสั่ง Step Speed 1 m/s จาก Computer ให้ LM629 แล้ววัดผลตอบสนองทางความเร็วได้ดังรูปที่ 4.5 จากผลตอบสนองทางความเร็วที่ได้ สามารถประมาณได้เป็นระบบอันดับสอง เราสามารถประมาณค่าพารามิเตอร์ต่างๆของระบบได้โดยปรับค่า ζ และ ω_n เพื่อให้ได้ผลตอบสนองที่ใกล้เคียงกับผลตอบสนองที่ได้จากระบบจริงมากที่สุด โดยพิจารณาจากกราฟผลตอบสนองของระบบจริงกับผลตอบสนองที่ได้จากการ Simulate บน MATLAB ซึ่งได้ผลดังนี้

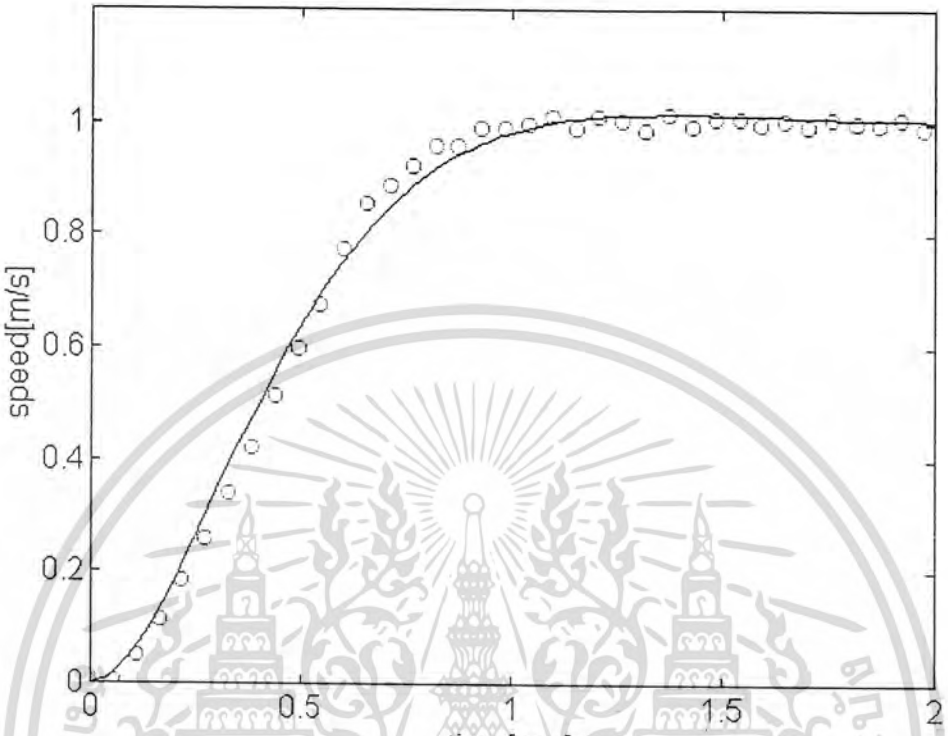
$$\frac{C(s)}{R(s)} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \qquad \text{----------(4.5)}$$

ω_n = 3.78 rad/sec

ζ = 0.8

C(s) = ความเร็วของตัวรถเครน

R(s) = คำสั่งความเร็วที่ส่งมาจาก Computer



รูปที่ 4.5 ooo ผลตอบสนองจริง

บทที่ 5

การ Simulation และการทดลอง

ในบทนี้จะแสดงถึงผลที่ได้จากการจำลองการทำงานของระบบ (Simulation) โดยคอมพิวเตอร์ และผลการทดลองที่ได้จากการทดลองจริงซึ่งมีทั้งแบบที่ใช้การประมาณค่าสเตทจากความสัมพันธ์ทางฟิสิกส์และแบบที่ใช้ตัวสังเกตสเตทที่ออกแบบโดยคาลมานฟิลเตอร์

5.1 การ Simulation

จาก สมการ(2.9 ในบทที่ 2)

$$\ddot{\theta} = -(\frac{g}{\ell})\theta - (\frac{\ddot{P}}{\ell}) \qquad \text{----------(5.1)}$$

และจาก สมการ(4.5 ในบทที่ 4)

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

เมื่อ $\theta(t)$ = มุมการแกว่งของภาระ g = ความเร่งเนื่องจากแรงโน้มถ่วงของโลก

$\chi(t)$ = ตำแหน่งของรถเครน ℓ = ความยาวของเชือกที่ใช้แขวนภาระ

$C(s)$ = ความเร็วของตัวรถเครน

$R(s)$ = คำสั่งความเร็วที่ส่งมาจากคอมพิวเตอร์

จากสมการที่ (5.1) และ (5.2)นำมาเขียนโมเดลของระบบ , $\dot{x} = Ax + Bu$, ได้ดังนี้

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

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{\chi} \\ \ddot{\chi} \\ \dddot{\chi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ -\frac{g}{\ell} & 0 & 0 & 0 & -\frac{1}{\ell} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\omega_n^2 & -2\zeta\omega_n \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ \chi \\ \dot{\chi} \\ \ddot{\chi} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \omega_n^2 \end{bmatrix} u$$

$$y = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ \chi \\ \dot{\chi} \\ \ddot{\chi} \end{bmatrix}$$

นำพารามิเตอร์ต่างๆของระบบเครนในหัวข้อที่ 4.3 ,$\omega_n = 3.78$ ζ=0.8 จากการประมาณ Transfer Function ของ LM629 กับ มอเตอร์ ซึ่งพารามิเตอร์ต่างๆมีค่าดังนี้

กำหนดให้เชือกที่ใช้แขวนโหลดมีความยาว l = 0.8 m นำพารามิเตอร์ต่างๆแทนในโมเดลของระบบได้ดังนี้

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{x} \\ \ddot{x} \\ \dddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ -12.2625 & 0 & 0 & 0 & -1.25 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -14.2884 & -6.048 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 14.2884 \end{bmatrix} u$$

จากนั้นทำการแปลง State Space Model เวลาต่อเนื่องให้เป็นเวลาดิสครีต ให้ sampling time = 0.1 sec โดยใช้ฟังชั่น c2d ในโปรแกรม MATLAB

สามารถหาค่า G และ H ของระบบดีสครีต , $x_{k+1} = Gx_k + Hu_k$, ได้ดังนี้

```
>> [G,H]=c2d(A,B,0.1)
G =  0.9393   0.0980        0    0.0025   -0.0050
    -1.2013   0.9393        0    0.0721   -0.0894
          0        0   1.0000    0.0980    0.0041
          0        0        0    0.9417    0.0733
          0        0        0   -1.0469    0.4985
H =  -0.0025
     -0.0721
      0.0020
      0.0583
      1.0469
```

กำหนดให้ ดัชนีการทำงานของ LQR เป็น

$$J = \sum_{k=0}^{\infty} [\, x^T(k)Qx(k) + u^T(k)Ru(k) \,]$$

คำนวณค่า LQR Steady State Gain Matrix , K โดยกำหนดให้

Q=diag{[100 1 1000 1 1]} R=[200]

นำค่า G,H,Q,R ที่ได้ไปคำนวนค่า K ใน MATLAB ได้ดังนี้

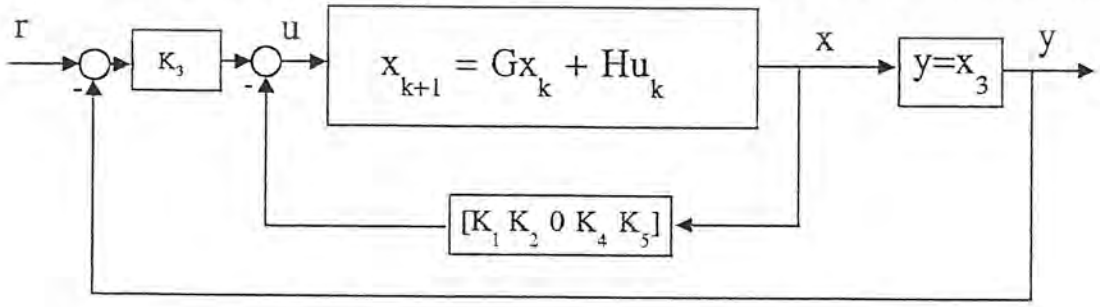
```
>> Q=diag([100 1 1000 1 1]); R=[200];
>> [K,S,E]=dlqr(G,H,Q,R)
K =[-0.3268   -0.1655    1.9859    0.8470    0.1546]
```

$K_1 = -0.3268$; $K_2 = -0.1655$; $K_3 = 1.9859$; $K_4 = 0.8470$; $K_5 = 0.1546$;

```
S =
  1.0e+003 *
    2.3825    0.0217   -0.7427   -0.8263   -0.0677
    0.0217    0.1776    0.1745    0.0936   -0.0232
   -0.7427    0.1745    9.3008    2.8552    0.3140
   -0.8263    0.0936    2.8552    1.2808    0.1357
   -0.0677   -0.0232    0.3140    0.1357    0.0240
E =
   0.7379 + 0.1828i
   0.7379 - 0.1828i
   0.7895
   0.9128 + 0.3334i
   0.9128 - 0.3334i
```

โดย E=EIG(G-HK) = Eigenvalue ของระบบจะเห็นได้ว่าระบบ ควบคุมวงปิดที่ออกแบบโดยวิธี LQR ข้างต้นมีเสถียรภาพเนื่องจากค่าโพลวงปิดที่ได้อยู่ภายในวงกลมหนึ่งหน่วย

โครงสร้างการทำงานของระบบควบคุมการเคลื่อนที่ และการแกว่งของโหลดที่แขวนบนรถเครนมีลักษณะเป็นการป้อนกลับสเตท โดยนำทุกๆสเตทที่วัดมาได้ในแต่ละคาบการชักตัวอย่างมาคูณค่าเกน K แล้วนำผลคูณที่ได้ป้อนกลับให้เป็นอินพุท u เพื่อควบคุมให้ทุกๆสเตทลู่เข้าสู่ศูนย์ยกเว้น x_3 ซึ่งเป็นตำแหน่งของรถเครนจะลู่เข้าสู่ค่าตำแหน่งอ้างอิง r นั่นก็คือ อินพุตจะลู่เข้าสู่ศูนย์เมื่อ r-y = 0 และ สเตททุกๆสเตทเป็นศูนย์โดยสามารถเขียน บล็อกไดอะแกรม การทำงานได้ดังนี้



รูปที่ 5.1 บล็อกไดอะแกรมการทำงานของระบบควบคุม

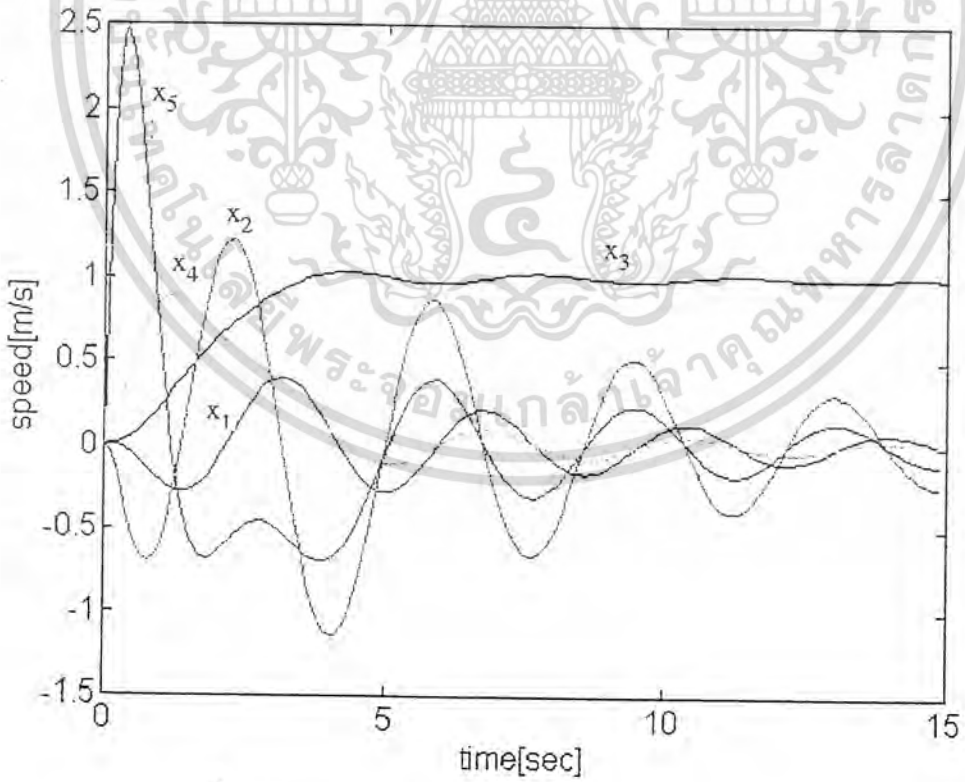
เราสามารถนำความสัมพันธ์ดัง บล็อกไดอะแกรม ข้างบนมาเขียนเป็นสเตทป้อนกลับรวมได้ดังนี้

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

นำผลที่ได้จากสมการข้างบนไป Simulate เพื่อหาผลตอบสนองต่อ Step Input (โดยสมมติว่าเราสามารถวัดค่าสเตทมาได้ทุกสเตท) ซึ่งสามารถทำได้ดังนี้

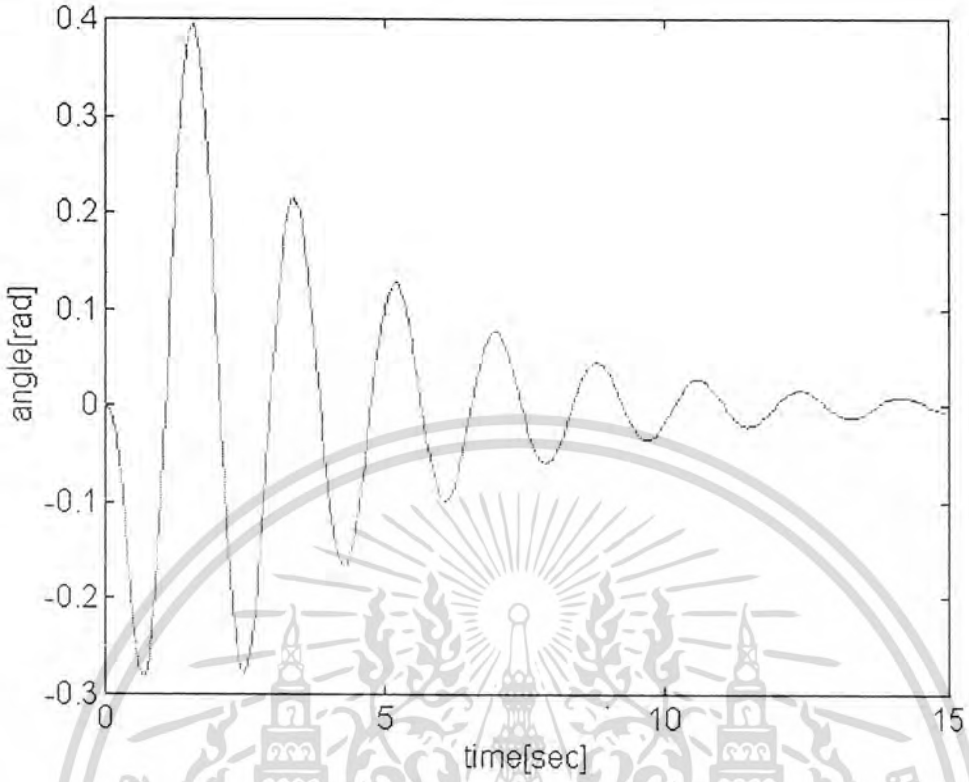
```
>>K =[-0.3268   -0.1655    1.9859    0.8470    0.1546]
>>GG=G-H*K;
>>HH=H*K(3);
>>CC=C; DD=D;
>>[y,x]=dstep(GG,HH,CC,DD);
```

ผลจากการ Simulate ข้างต้นจะได้ผลตอบสนองของตำแหน่งรถเครนดังรูปที่ 5.2ง ผลตอบสนองของมุมการแกว่งดังรูปที่ 5.2ข ผลตอบสนองของสเตทต่างๆดังรูปที่ 5.2ก โดยจะเห็นได้ว่าทุกๆ สเตท จะลู่เข้าสู่ศูนย์ยกเว้นเอาท์พุตของระบบที่จะลู่เข้าสู่ค่าตำแหน่งอ้างอิง

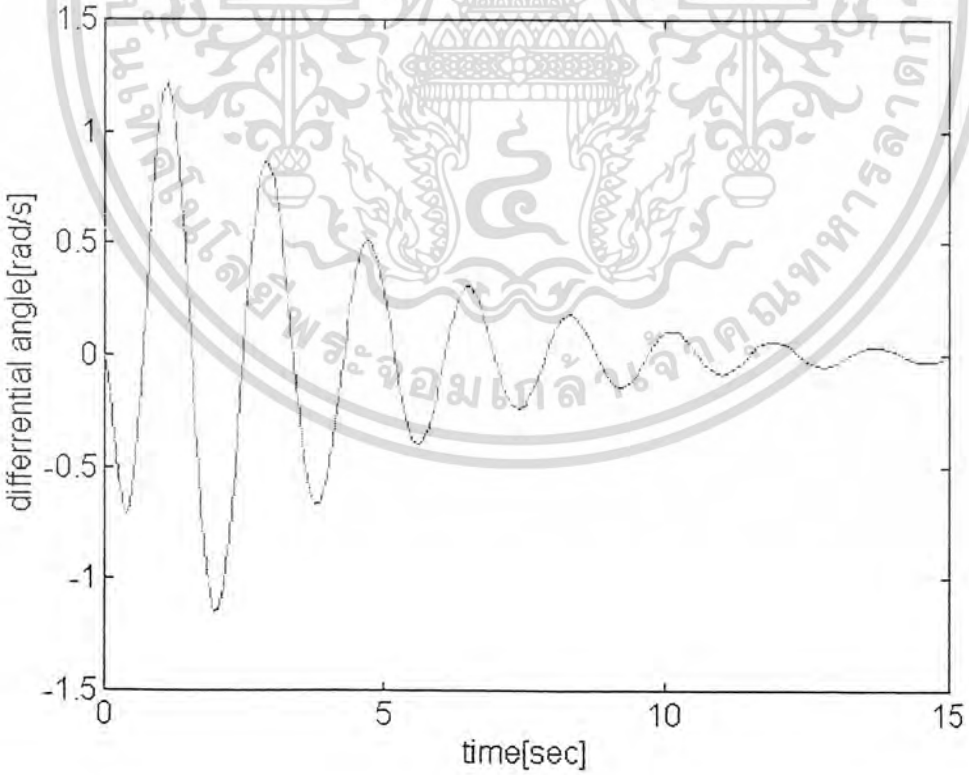


รูปที่ 5.2ก ผลตอบสนองของ สเตท ต่างๆของระบบเครน

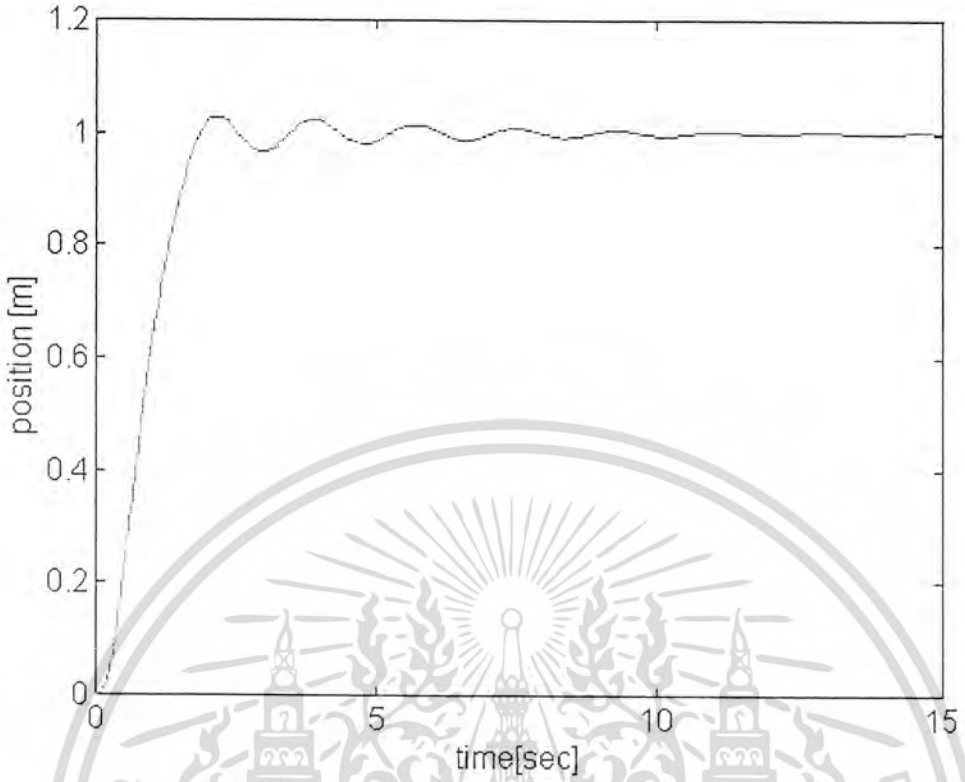
เมื่อ $x_1 = \theta$, $x_2 = \dot{\theta}$, $x_3 = \chi$, $x_4 = \dot{\chi}$, $x_5 = \ddot{\chi}$



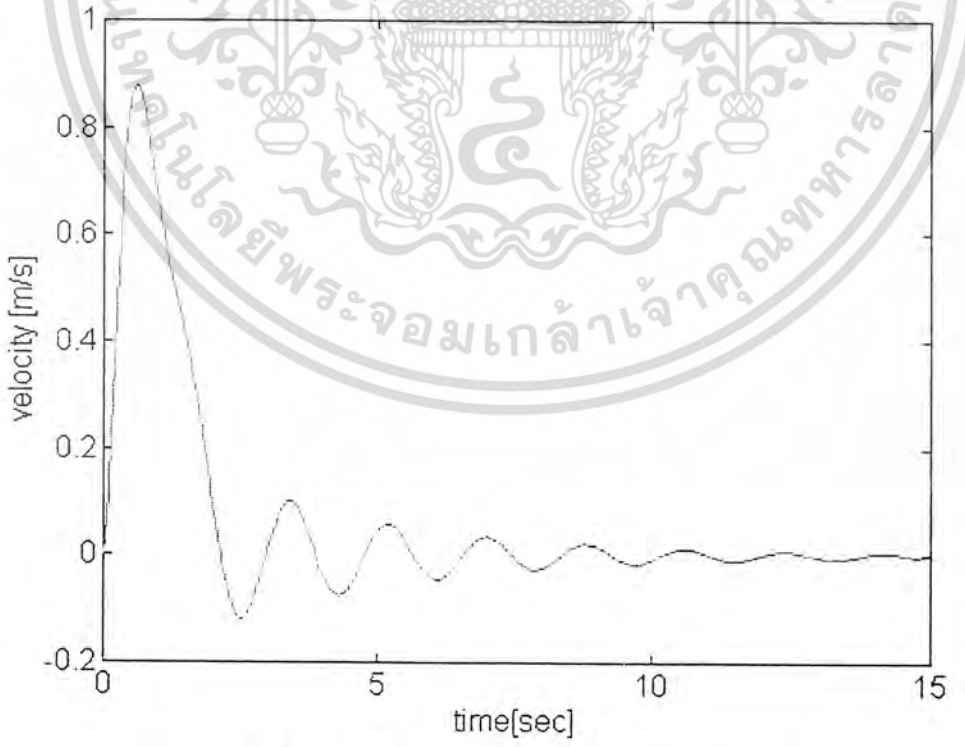
รูปที่ 5.2ข ผลตอบสนองมุมการแกว่ง



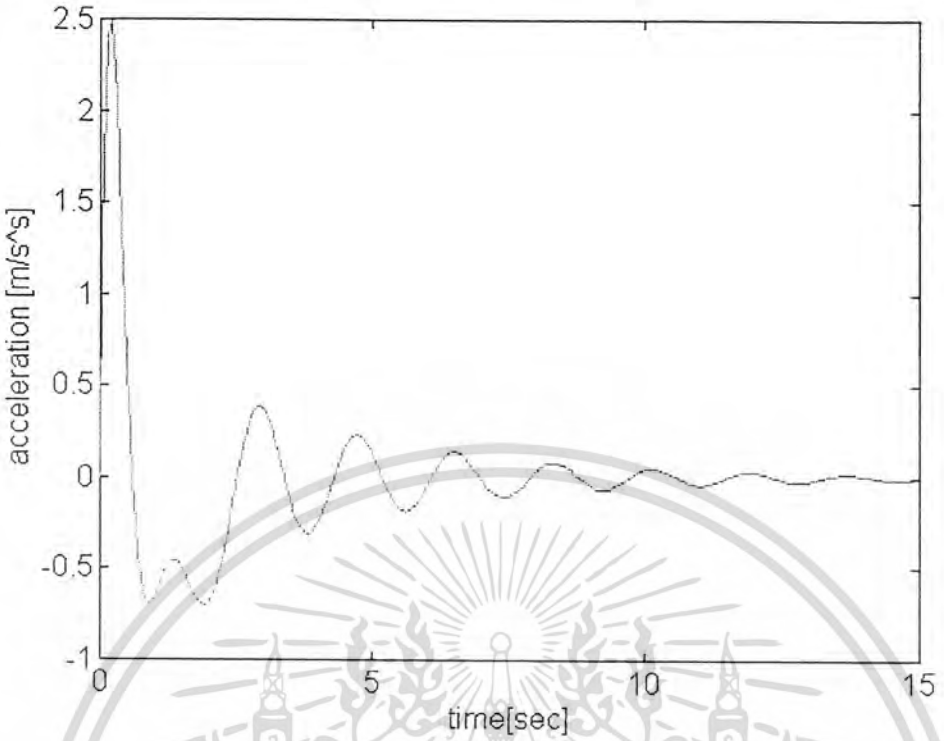
รูปที่ 5.2ค ผลตอบสนองความเร็วเชิงมุม



รูปที่ 5.2ง ผลตอบสนองตำแหน่งของรถเครน



รูปที่ 5.2จ ผลตอบสนองความเร็วของรถเครน



รูปที่ 5.2จ ผลตอบสนองความเร่งของรถเครน

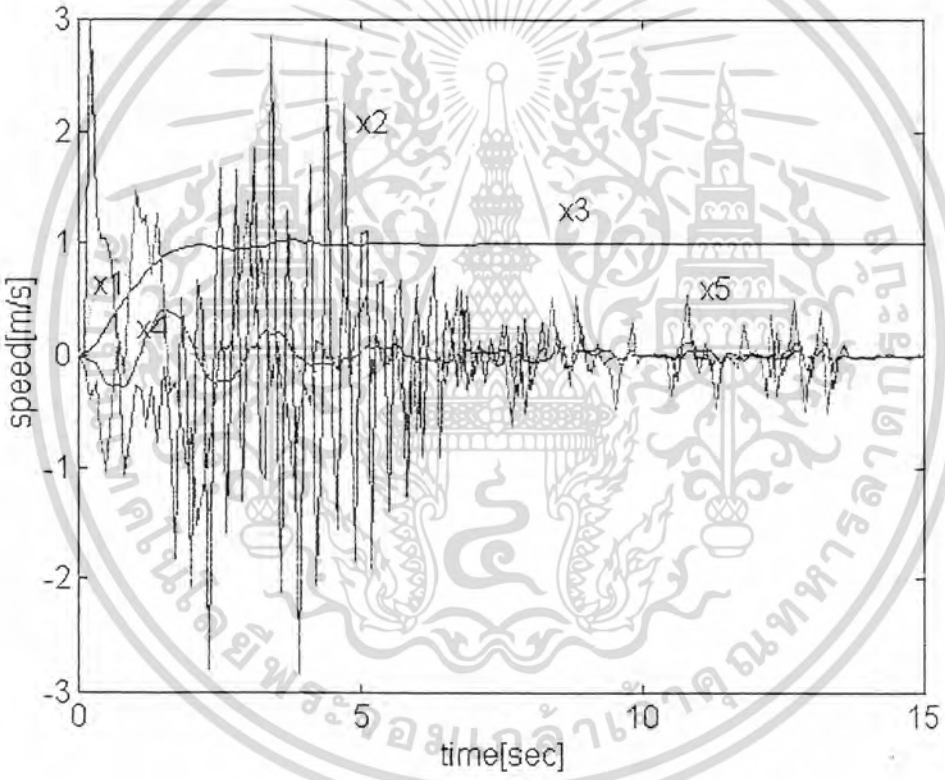
5.2 การทดลอง

5.2.1 การทดลองเมื่อใช้การประมาณค่าสเตทจากความสัมพันธ์ทางฟิสิกส์

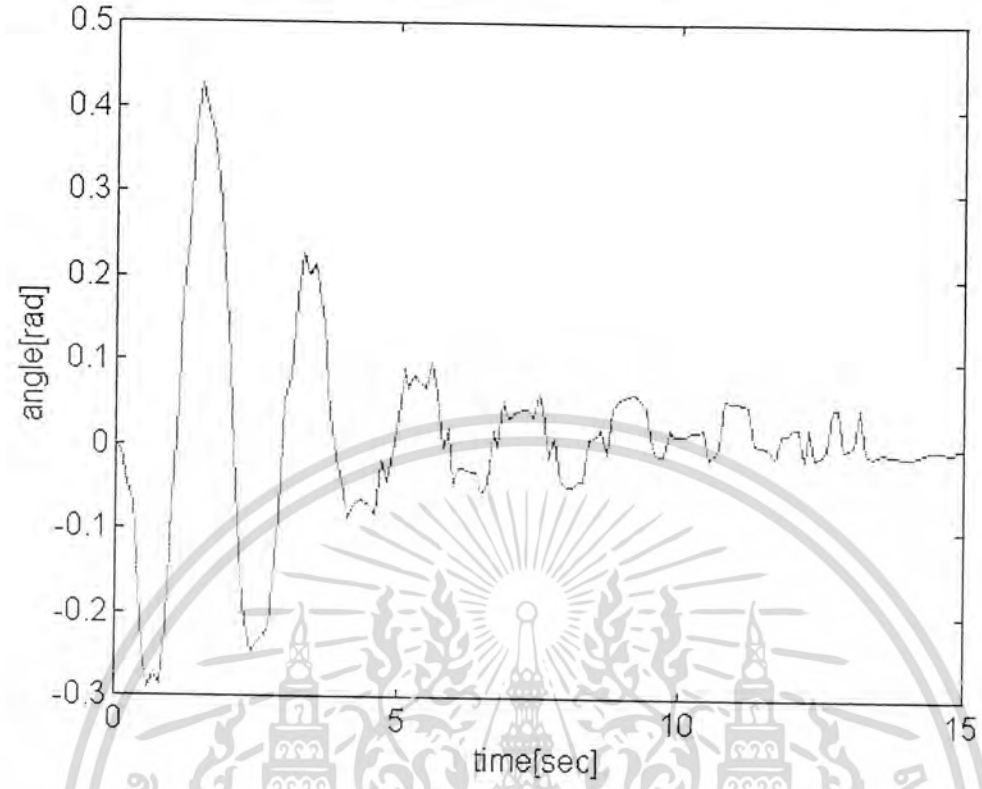
เมื่อทำการทดลองการควบคุมโดยใช้โปรแกรมควบคุมที่ใช้การประมาณค่าสเตทจากความสัมพันธ์ทางฟิสิกส์ และใช้ค่าเกนป้อนกลับ K ที่ออกแบบได้จาก MATLAB จากหัวข้อที่ 5.1 มาคูณกับค่าสเตทต่างๆทั้งที่ได้จากการวัดจริงและได้จากการคำนวณ ตามวิธี LQR เพื่อหาค่าสัญญาณควบคุม (ดู source code ของโปรแกรมควบคุมชื่อ LQR . C ได้ในภาคผนวก)

จะได้ผลการทดลองรูปที่ 5.3ก-5.3ฉ

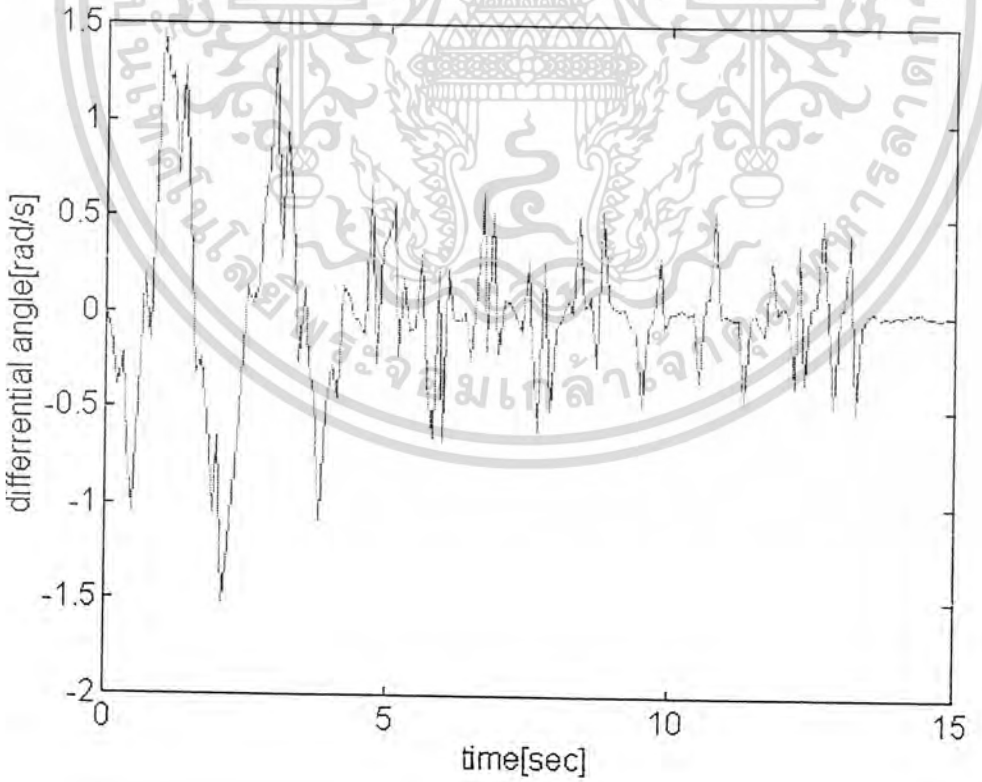
เมื่อ $x_1 = \theta$, $x_2 = \dot{\theta}$,$x_3 = \mathcal{X}$, $x_4 = \dot{\mathcal{X}}$, $x_5 = \ddot{\mathcal{X}}$



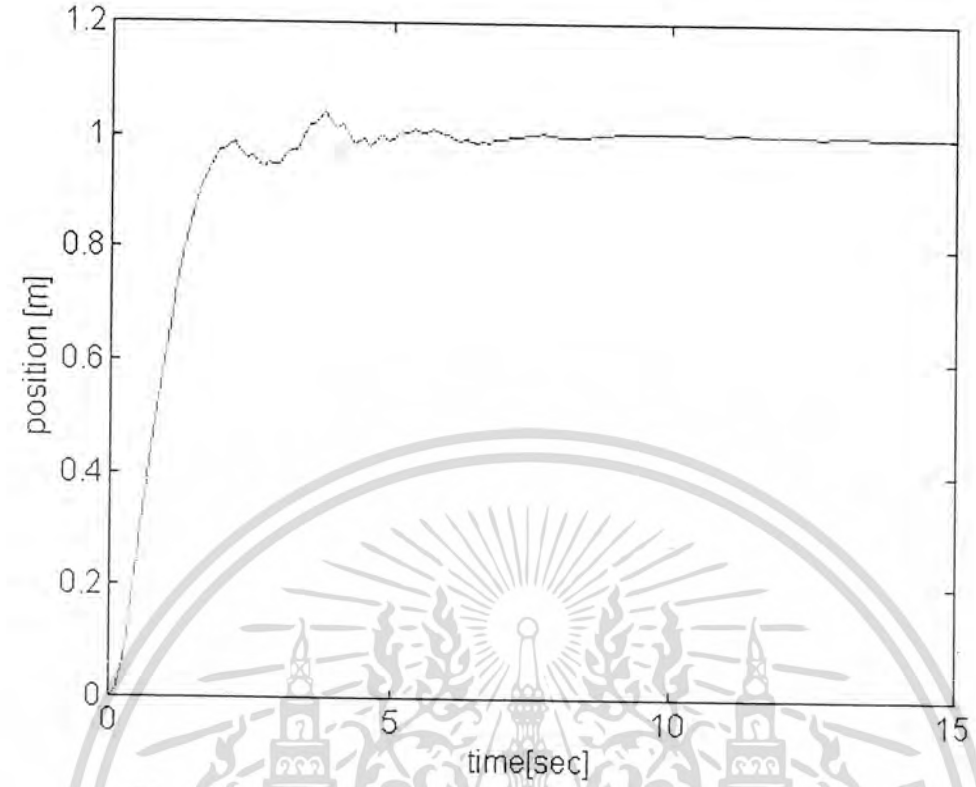
รูปที่ 5.3ก ผลตอบสนองที่ได้จากการทดลอง



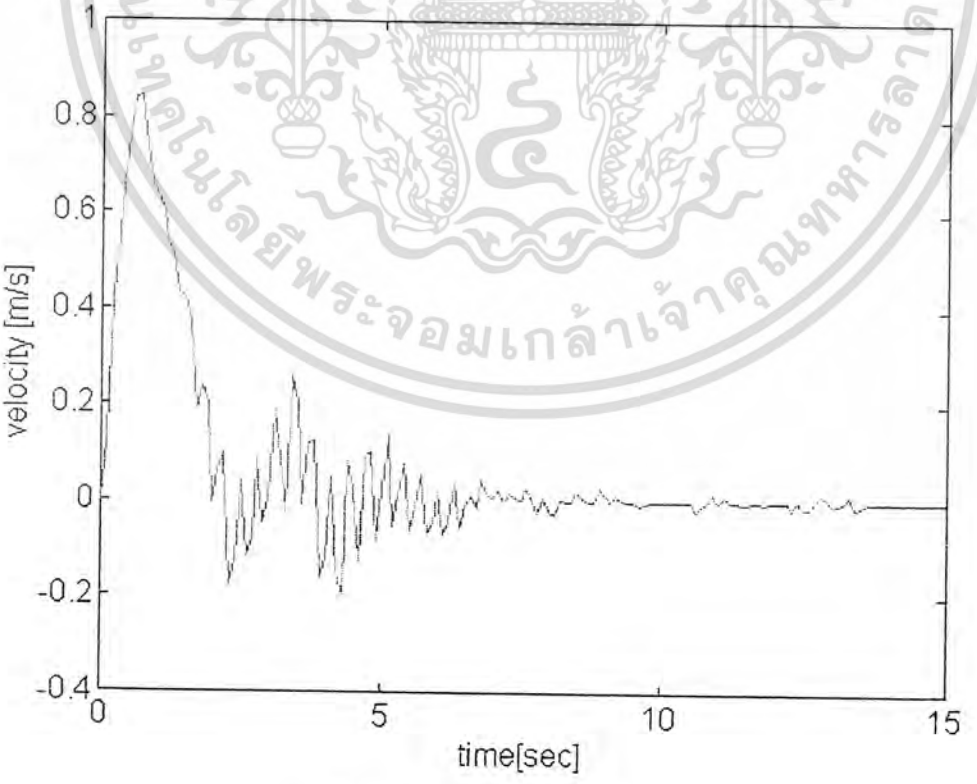
รูปที่ 5.3ข ผลตอบสนองของมุมการแกว่ง



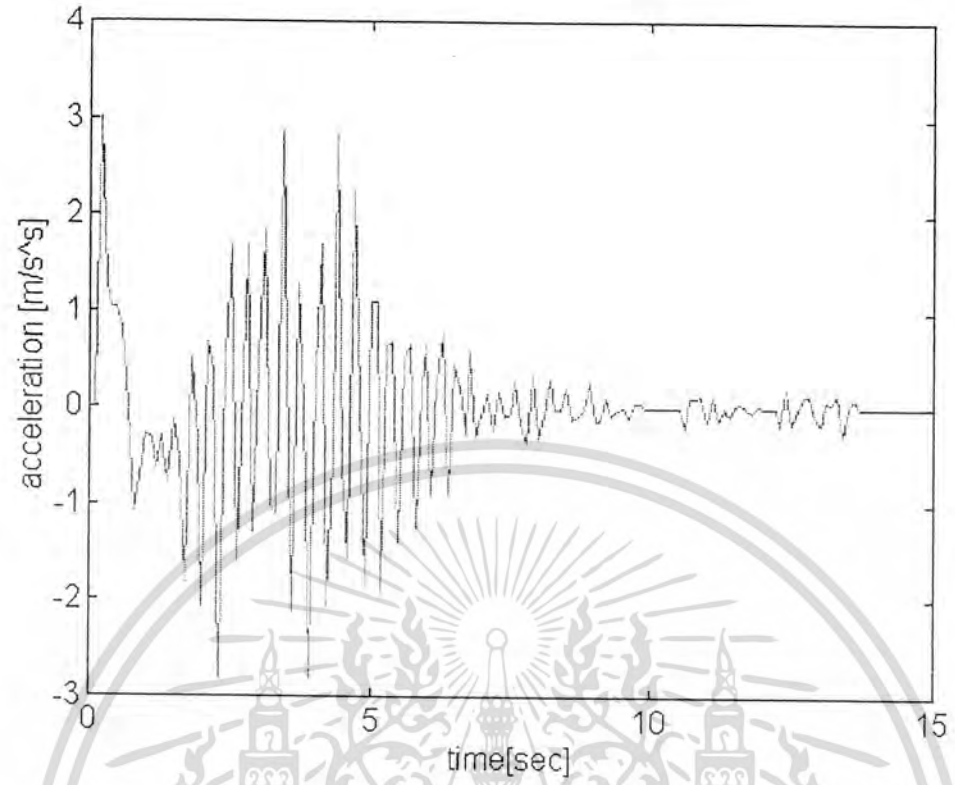
รูปที่ 5.3ค ผลตอบสนองของความเร็วเชิงมุม



รูปที่ 5.3ง ผลตอบสนองของตำแหน่ง

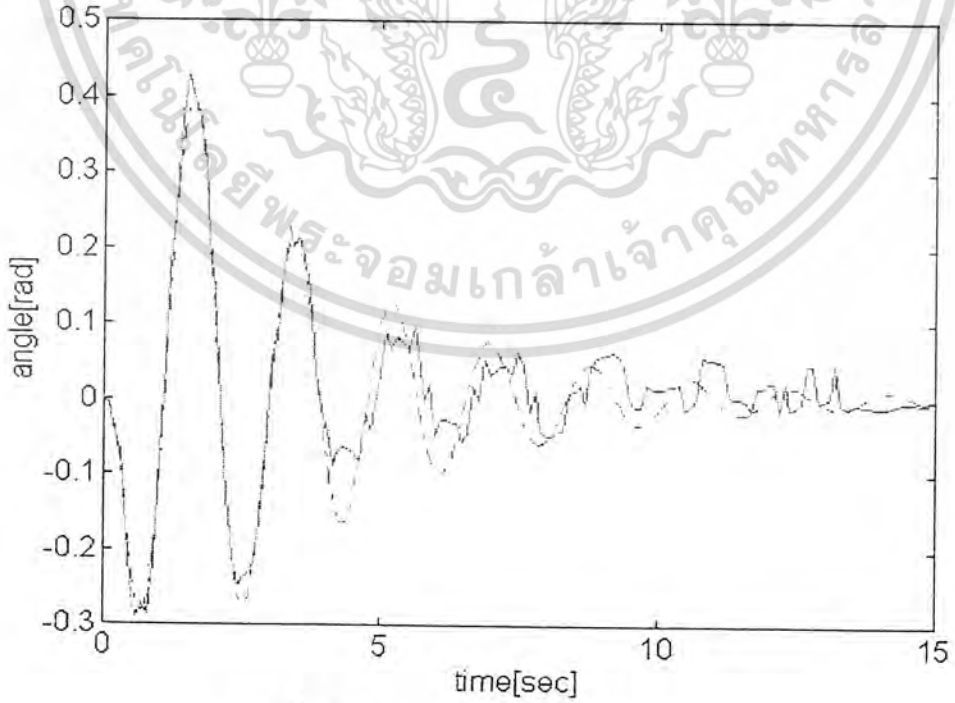


รูปที่ 5.3จ ผลตอบสนองของความเร็ว

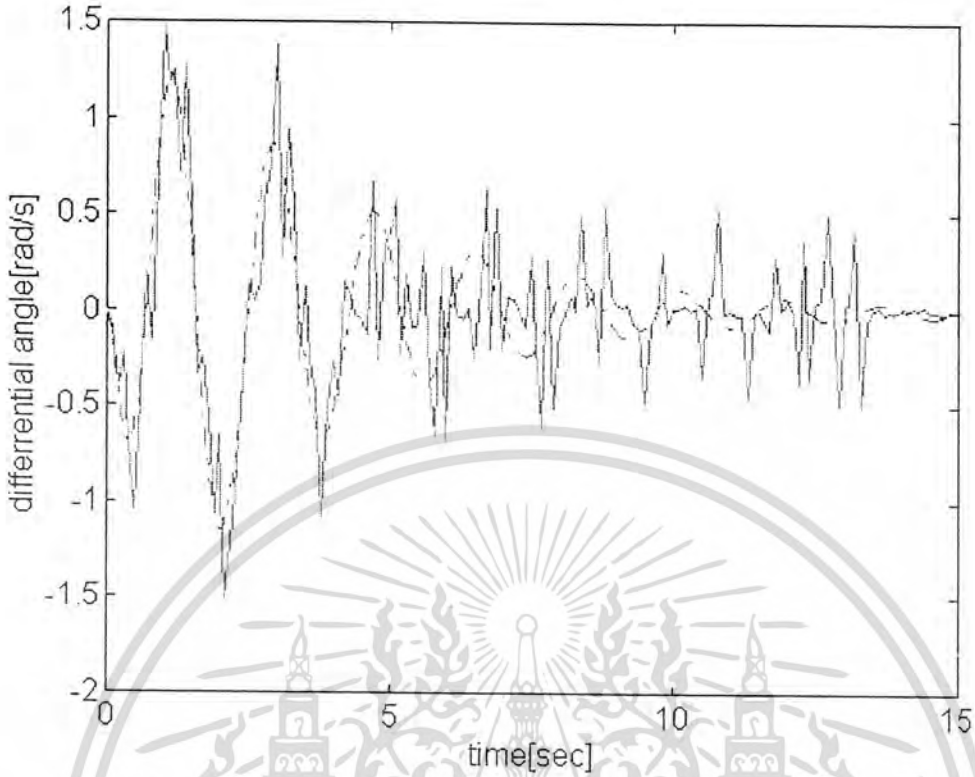


รูปที่ 5.3ฉ ผลตอบสนองของความเร่ง

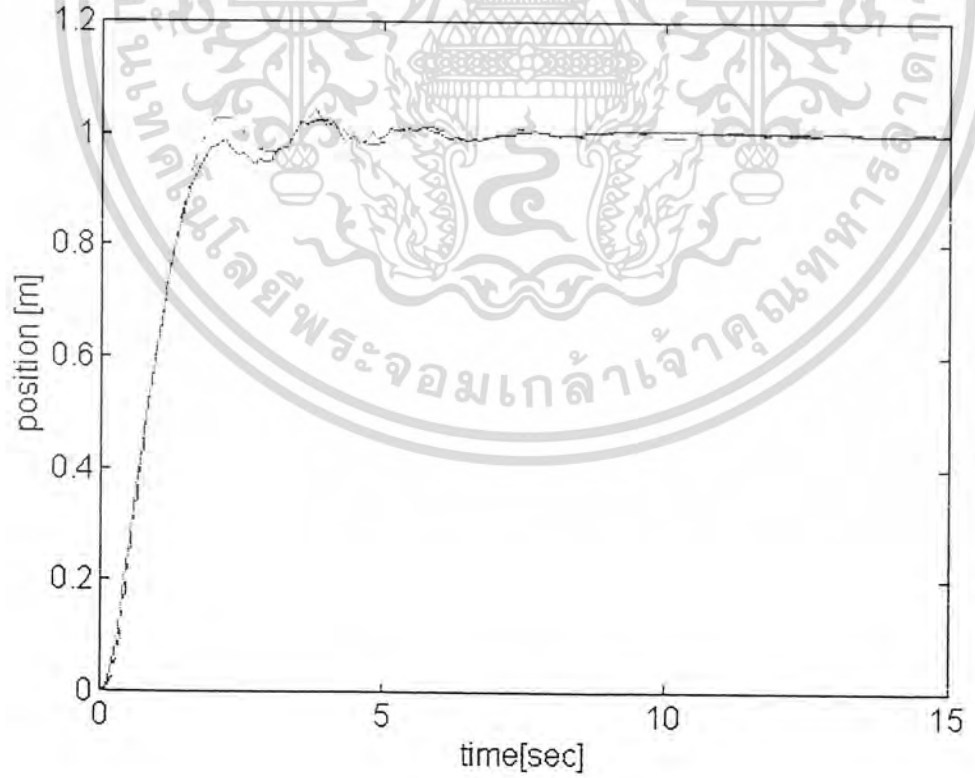
5.2.2 ผลตอบสนองของเครนที่ได้จากการ Simulate จากหัวข้อ 5.1 เปรียบเทียบกับผลตอบสนองที่ได้การทดลองทดลองที่ได้จากหัวข้อ 5.2.1 โดยเส้นทึบแสดงถึงผลตอบสนองที่ได้จากการทดลอง



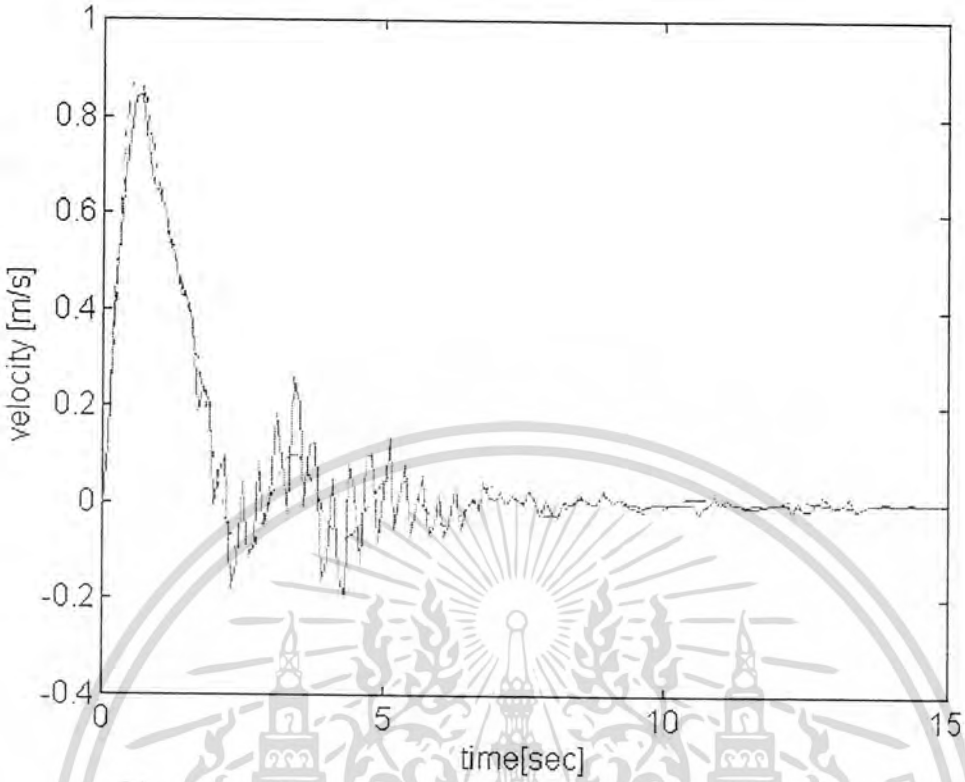
รูปที่ 5.4ก ผลตอบสนองของมุมการแกว่ง



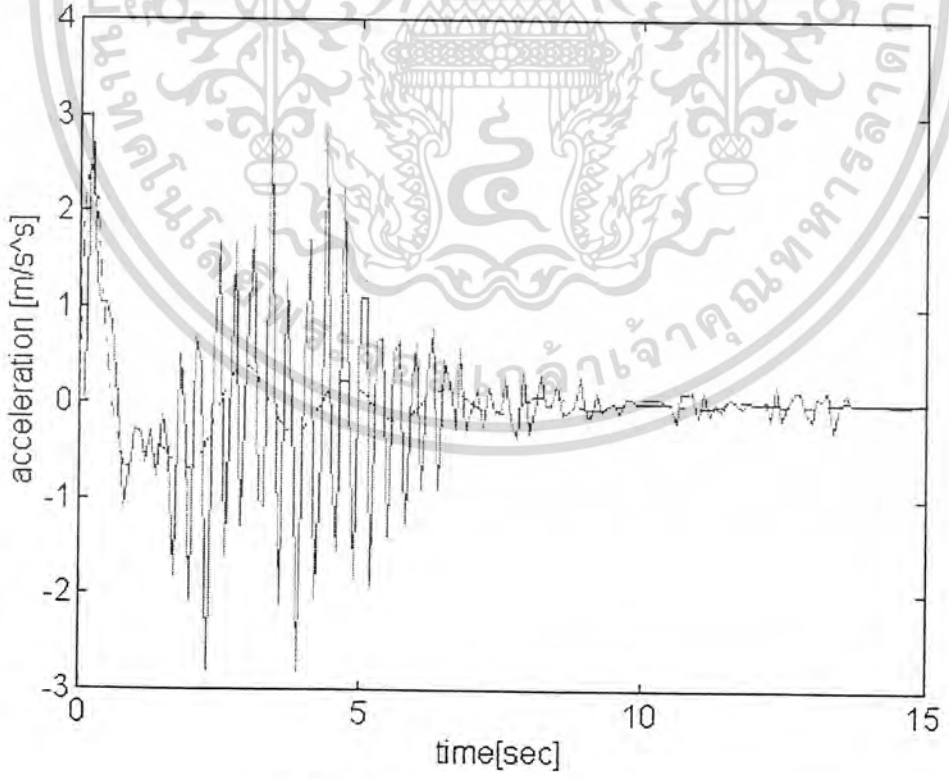
รูปที่ 5.4ข ผลตอบสนองของความเร็วเชิงมุม



รูปที่ 5.4ค ผลตอบสนองของตำแหน่ง



รูปที่ 5.4ง ผลตอบสนองของความเร็ว



รูปที่ 5.4จ ผลตอบสนองของความเร่ง

จากผลการทดลองตามรูปที่ 5.4ก ถึง 5.4จ นั้นจะเห็นได้ว่าผลตอบสนองที่ทดลองได้จริงกับที่ได้จากการ Simulate ของมุมการแกว่งและตำแหน่งรถเครนใกล้เคียงกันมาก ส่วนความเร็วเชิงมุม ความเร็วและความเร่ง ค่าที่ได้จากการทดลองนั้นมีความแปรปรวนจากค่าที่ Simulate ได้อยู่สูงแต่ค่าที่เป็นส่วนหลัก(Fundamental) นั้นก็ยังใกล้เคียงกับค่า Simulate ทั้งนี้อาจเกิดจากสัญญาณรบกวนความถี่สูงซึ่งเข้ามาที่เอาท์พุตของระบบทำให้เกิดการเปลี่ยนขนาดเล็กๆที่เอาท์พุตในช่วงเวลาเล็กๆทำให้เมื่อเราหาสเตทที่เหลือจากเอาท์พุต(เช่น สเตทความเร็วจากการนำตำแหน่งใน sampling time นี้ลบกับตำแหน่งในsampling time ที่แล้ว นำมาหารด้วยค่า sampling time)จะมีความแปรปรวนสูง ,เนื่องจาก sampling time มีค่าเล็กมาก, แต่มีค่าที่เป็นส่วนหลัก(Fundamental) เท่ากับค่าความเร็วที่ Simulate ซึ่งในหัวข้อถัดไปเราจะนำวิธีคาลมานฟิลเตอร์มาใช้ในการประมาณสเตท เพื่อแก้ปัญหาความแปรปรวนของสเตทที่หาได้จากความสัมพันธ์ทางฟิสิกส์

5.2.3 การทดลองเมื่อใช้การประมาณค่าสเตทจากตัวสังเกตสเตทที่ออกแบบโดยคาลมานฟิลเตอร์

เมื่อทำการทดลองควบคุมโดยใช้โปรแกรมควบคุมที่ทำการประมาณค่าสเตทจากตัวสังเกตสเตทที่ออกแบบโดยคาลมานฟิลเตอร์ซึ่งเป็นแบบไม่แปรค่ากับเวลา และใช้ค่าเกนป้อนกลับ K ที่ออกแบบได้จากจากหัวข้อที่ 5.1 นำมาคูณกับค่าสเตทต่างๆที่ได้จากคาลมานฟิลเตอร์ เพื่อหาค่าสัญญาณควบคุม (ดู source code ของโปรแกรมควบคุมชื่อ LQG .C ได้ในภาคผนวก) เราเรียกระบบควบคุมแบบนี้ว่าระบบ LQG

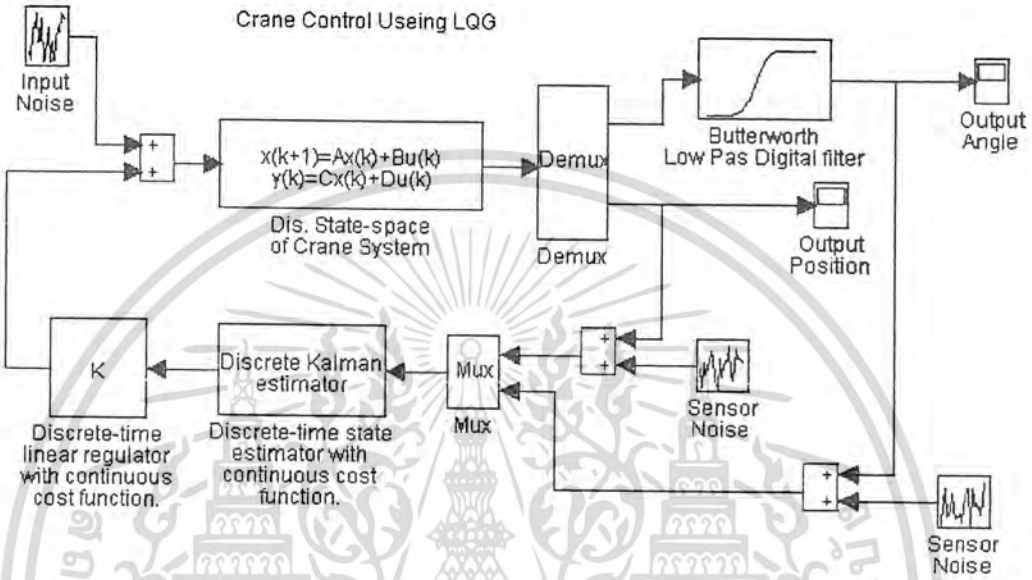
การทำงานของคาลมานฟิลเตอร์นั้นได้แสดงไว้ในบทที่ 3 โดยกำหนดให้ G เป็น Identity Matrix , Q_f^c = diag([100 100 100 100 100]) , R_f^c = [0.01 0.01] , $C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$

ดังนั้นเราสามารถหาค่า P ที่สภาวะคงตัวได้ดังสมการที่ (3.25) และเมื่อแทนค่า P ที่ได้ลงในสมการที่ (3.26) จะสามารถหาค่า Kalman Gain เพื่อใช้ในการสร้างสเตทดังสมการที่ (3.26) โดยขั้นตอนการทำงานของคาลมานฟิลเตอร์นั้นเป็นดังรูปที่ (3.4) แต่ค่า Kalman Gain ที่ใช้ในการทดลองนี้ไม่แปรค่ากับเวลา และใช้ค่าเวลาการชักตัวอย่างเท่ากับ 0.1 sec

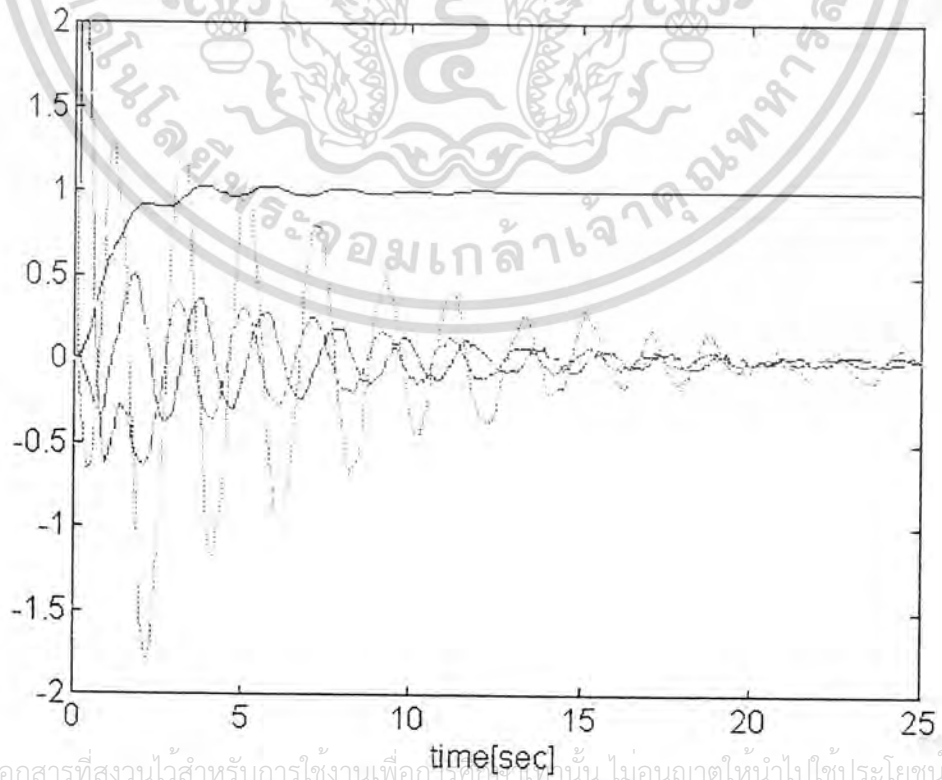
จากรูปที่ 5.5 จะเห็นได้ว่าระบบ LQG ที่ได้มีเอาท์พุท 2 ตัวคือมุมการแกว่งของภาระ(θ) และ ตำแหน่งของรถเครน(χ) จากนั้นนำเอาท์พุททั้งสองตัวและอินพุท(u)จะถูกป้อนให้คาลมานฟิลเตอร์เพื่อสร้างสเตทที่ต้องการ แล้วนำสเตทที่ได้คูณกับค่า Gain K ของ LQR เพื่อป้อนกลับเป็นอินพุทของระบบต่อไป

จากการทดลองในครั้งนี้เราได้พบว่า มีสัญญาณรบกวนความถี่ต่ำมากๆ เข้ามาที่อุปกรณ์วัดมุมทำให้ค่าของมุมที่วัดได้ไม่เป็นศูนย์ที่สภาวะคงตัว ซึ่งถ้าเรานำมุมที่ได้นี้ไปสร้างสเตทโดยใช้คาลมาลฟิลเตอร์แล้ว ไม่เฉพาะมุมที่ได้จะมีความผิดพลาดเท่านั้นสเตทอื่นๆเช่นความเร็วที่สภาวะคงตัวก็ผิดพลาดไปด้วย เนื่องจากสเตทที่คำนวนได้จากคาลมาลฟิลเตอร์ประกอบด้วยทั้ง เทอมที่มาจากโมเดลและเทอมที่ได้จากเอาท์พุทของระบบ ดังนั้นถ้าคาลมาลฟิลเตอร์คิดว่าระบบยังมีมุมที่สภาวะคงตัว สเตทความเร็วที่ประมาณได้จะยังมีค่าอยู่ทั้งๆที่ระบบจริงรถเครนหยุดนิ่งแล้ว ดังนั้นเราจึงแก้ปัญหานี้ได้โดยใช้ดิจิตอลฟิลเตอร์กรองเอาความถี่ต่ำออกจากมุมที่วัดได้ก่อนที่จะป้อนให้คาลมาลฟิลเตอร์ ซึ่งจะทำให้ค่าสเตทที่ประมาณได้มีความถูกต้อง

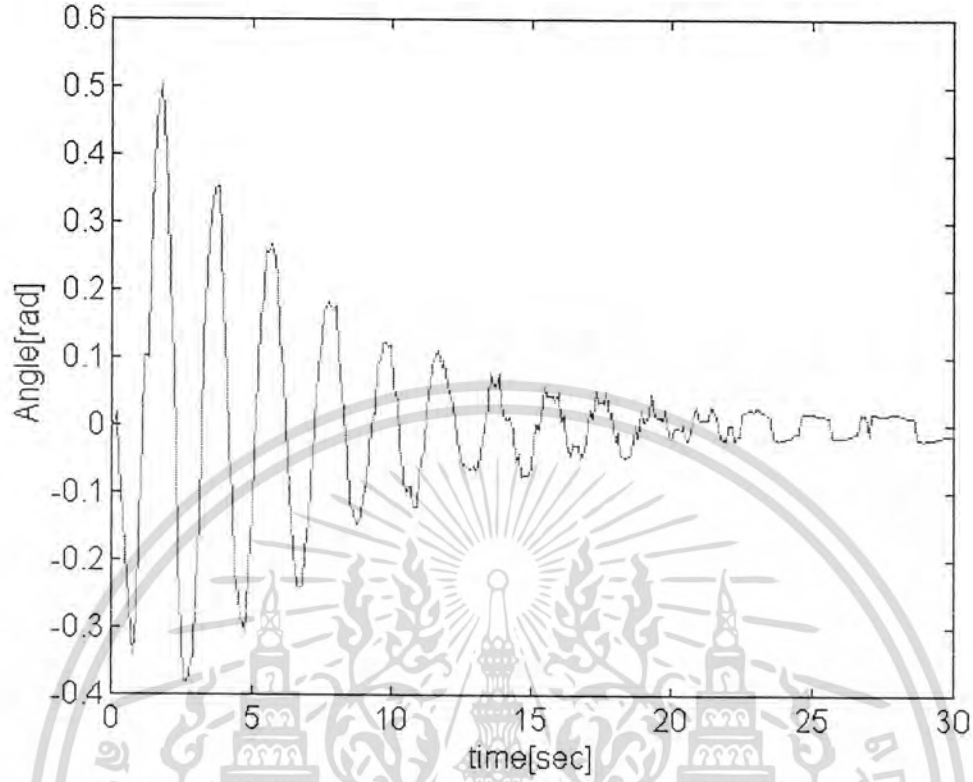
โครงสร้างการทำงานของระบบควบคุมที่ใช้คาลมาลฟิลเตอร์ซึ่งใช้ในการทดลองนี้แสดงดังรูปที่ 5.5 จะเห็นว่าได้มีการใช้ Digital Low Pass Filter เพื่อกรอง noise ความถี่ต่ำออกจากค่ามุมที่วัดได้ก่อนที่จะส่งต่อไปให้คาลมานฟิลเตอร์เพื่อนำเอาท์พุทไปประมาณค่าสเตทต่อไป ซึ่งจากการทดลองจะได้ผลตามรูปที่ 5.6ก-5.6ฉ เมื่อ $x_1 = \theta$, $x_2 = \dot{\theta}$, $x_3 = \chi$, $x_4 = \dot{\chi}$, $x_5 = \ddot{\chi}$



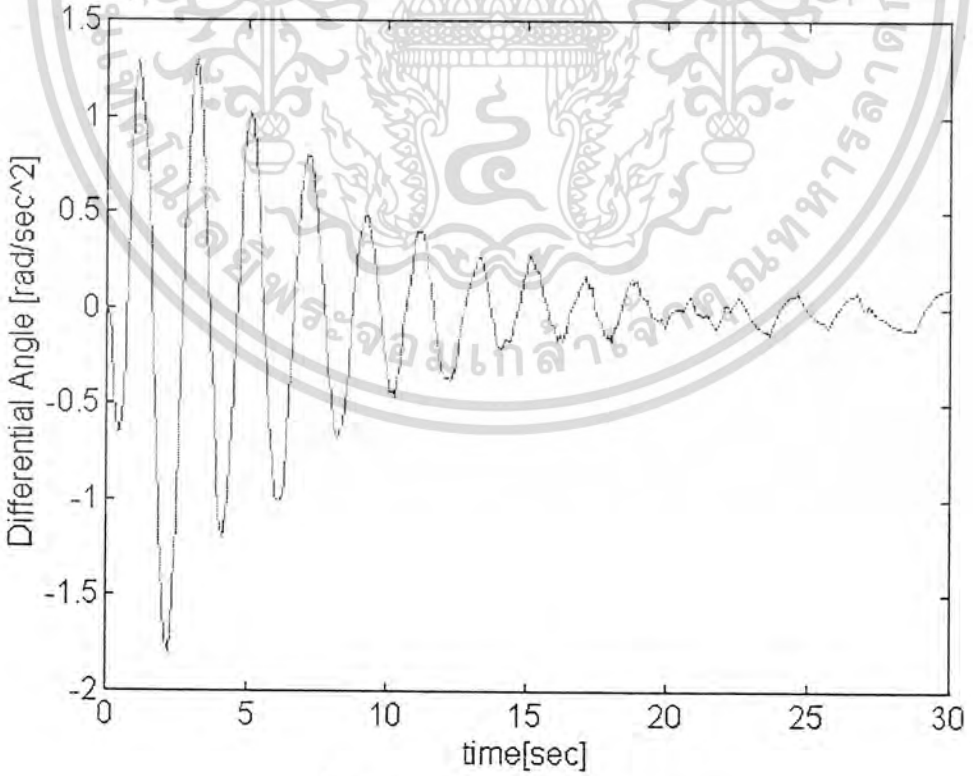
รูปที่ 5.5



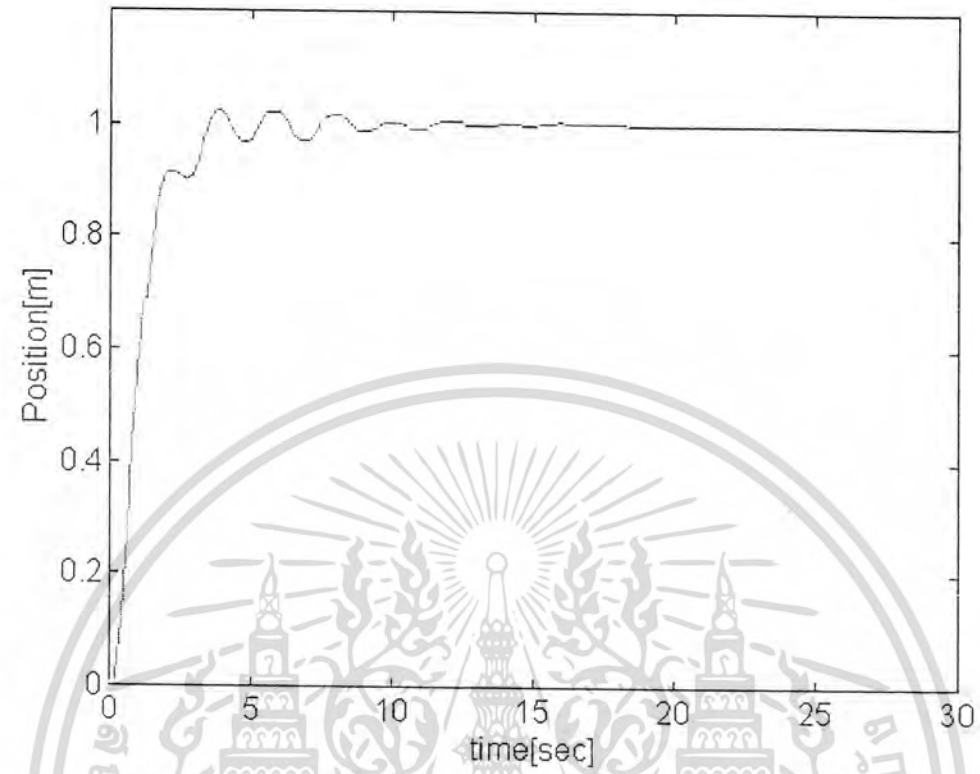
รูปที่ 5.6ก แสดงถึงสเตทต่างๆของระบบที่ควบคุมแบบ LQG



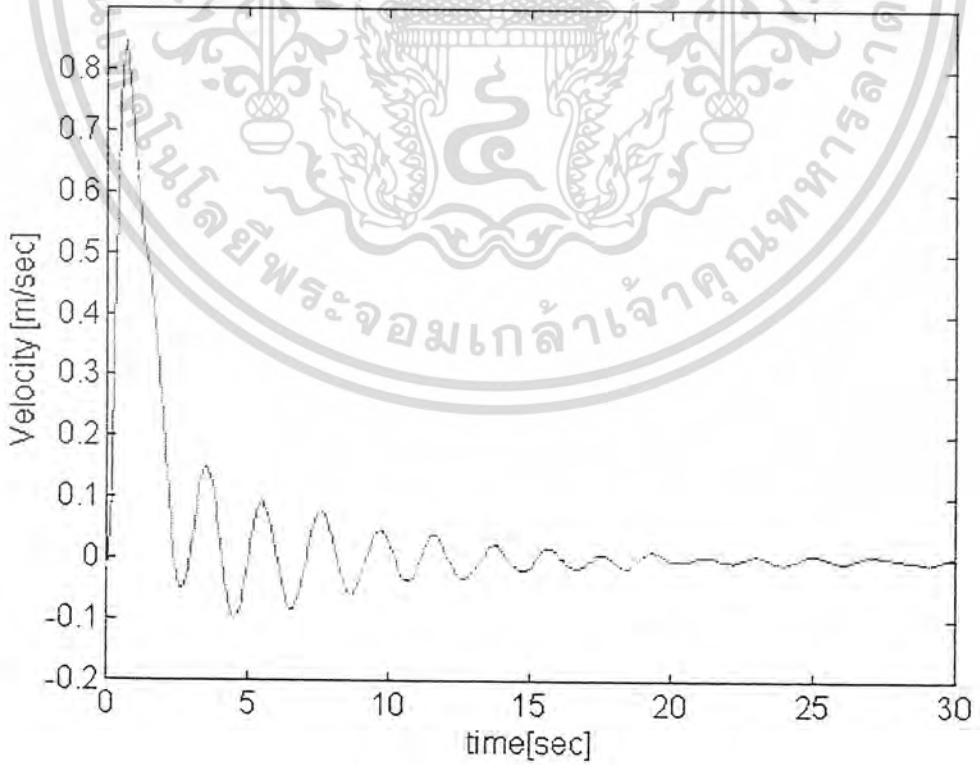
รูปที่ 5.6ข แสดงมุมการแกว่งของภาระ



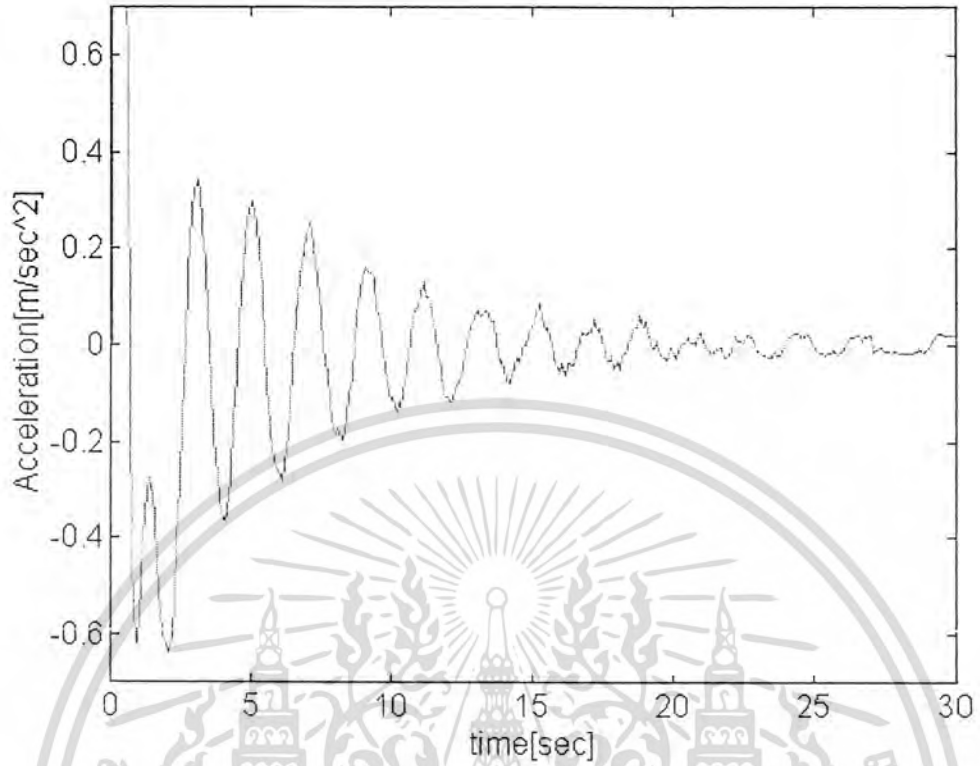
รูปที่ 5.6ค แสดงความเร็วเชิงมุมของภาระ



รูปที่ 5.6ง แสดงตำแหน่งของรถเครน

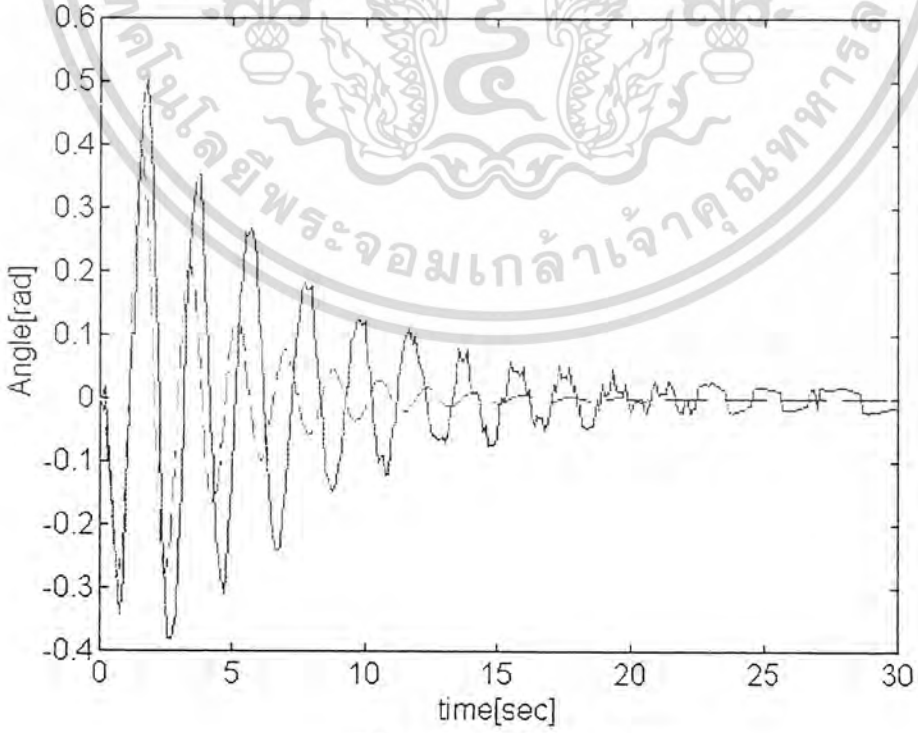


รูปที่ 5.6จ แสดงความเร็วของรถเครน

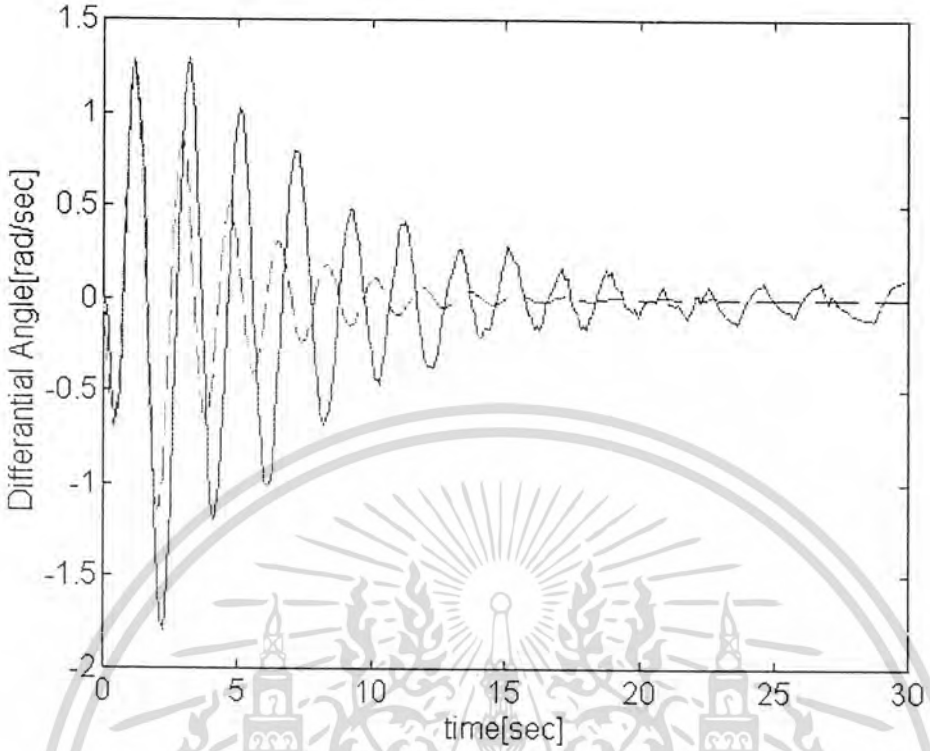


รูปที่ 5.6ฉ แสดงความเร่งของรถเครน

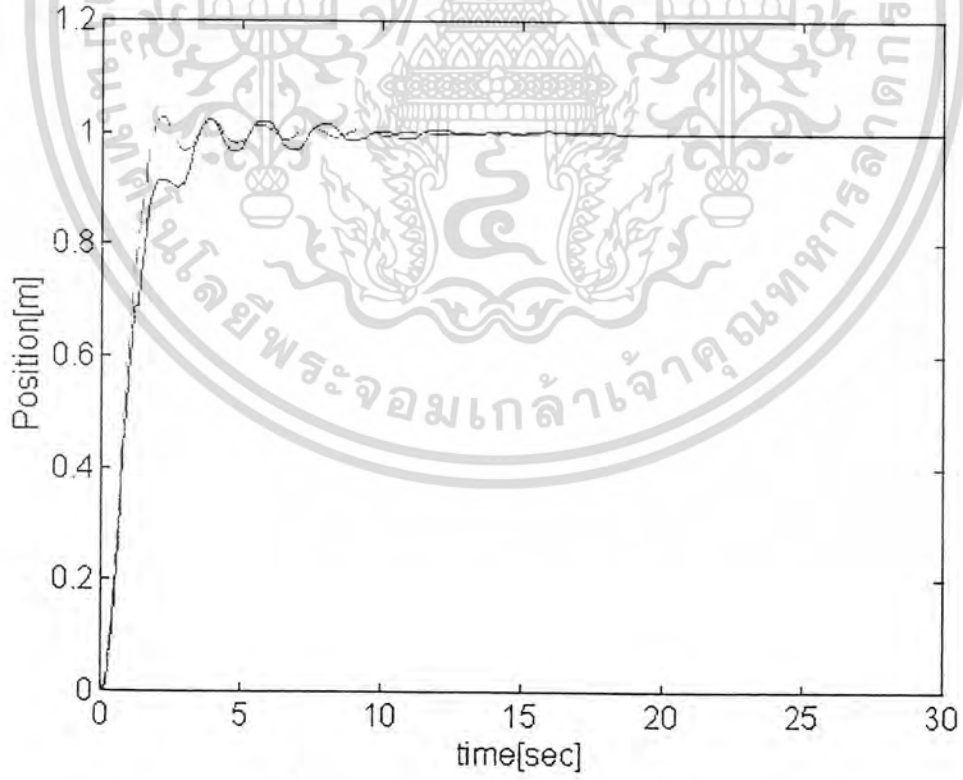
5.2.4 ผลตอบสนองของเครนที่ได้จากการ Simulate(แสดงด้วยเส้นประ) จากหัวข้อ 5.1 เปรียบเทียบกับผลตอบสนองที่ได้การทดลองทดลองที่ได้จากหัวข้อ 5.2.3(แสดงด้วยเส้นทึบ)



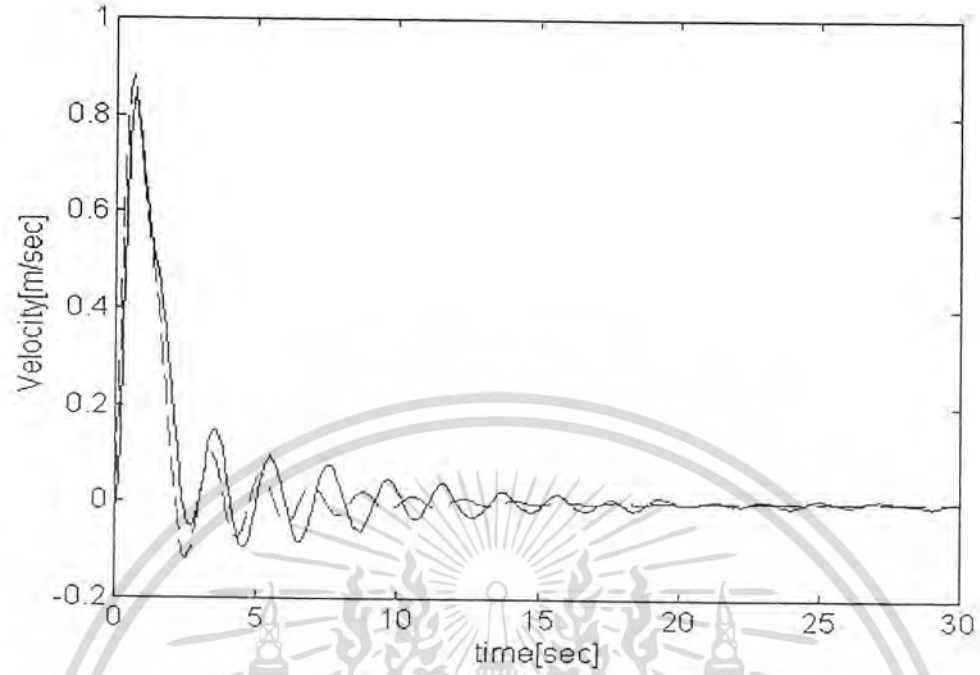
รูปที่ 5.7ก ผลเปรียบเทียบของมุม



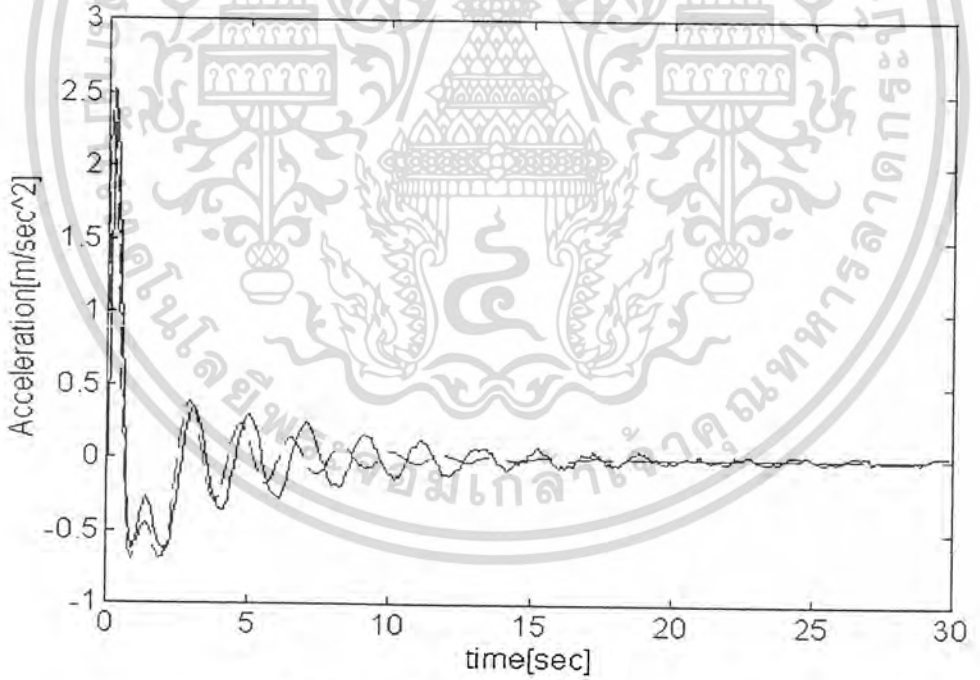
รูปที่ 5.7ข ผลเปรียบเทียบของความเร็วเชิงมุม



รูปที่ 5.7ค ผลเปรียบเทียบของตำแหน่งรถเครน



รูปที่ 5.7ง ผลเปรียบเทียบของความเร็วรถเครน



รูปที่ 5.7จ ผลเปรียบเทียบของความเร่งรถเครน

จากผลการทดลองจะเห็นว่าผลตอบสนองที่ได้นั้นมีความแปรปรวนน้อยลงคือผลตอบสนองที่ได้เรียบขึ้นกว่าการทดลองที่ 5.2 แต่มุมของภาระจะหยุดแกว่งช้ากว่าการทดลองที่ 5.2 ซึ่งใช้การประมาณสเตทโดยอาศัยความสัมพันธ์ทางฟิสิกส์ และผลตอบสนองของระบบที่ใช้คาลมานฟิลเตอร์นี้ก็ยังขึ้นอยู่กับการกำหนด ค่าความแปรปรวนของ process noise:Q_f และ measurement noise:R_f ซึ่งถ้าออกแบบค่าเหล่านี้ให้ถูกต้องขึ้นก็จะทำให้ผลตอบสนองที่ได้ดีขึ้น

บทที่ 6

ผลสรุปและข้อเสนอแนะของระบบควบคุมการเคลื่อนที่ของเครน

6.1 สรุปผลการทดลองการควบคุมการเคลื่อนที่ของเครน

1.การออกแบบระบบควบคุมการเคลื่อนที่ของเครนแบบ LQR นั้นลักษณะของผลตอบสนองที่ได้จะขึ้นอยู่กับความต้องการของผู้ออกแบบว่าต้องการให้ผลตอบสนองของแต่ละสเตทมีลักษณะลู่เข้าเร็วหรือช้าอย่างไร โดยกำหนดที่พารามิเตอร์ Q และ R ที่ใช้ในการออกแบบระบบควบคุมแบบ LQR

2.การทำงานของระบบควบคุมการเคลื่อนที่ของเครนแบบ LQG นั้นอาจสามารถแบ่งพิจารณาได้เป็นสองส่วนคือ ส่วนที่เป็น State Feedback ซึ่งออกแบบโดยวิธี LQR และ ส่วนคาลมานฟิลเตอร์ที่ใช้สร้างสเตท ซึ่งการออกแบบพารามิเตอร์ Q_f และ R_f ของคาลมานฟิลเตอร์นั้นจะต้องพิจารณาถึงรายละเอียดของแต่ละระบบ ว่าค่าเอาท์พุทที่วัดได้และสเตทต่างๆของระบบมีขนาดความแปรปรวนเป็นอย่างไร โดยค่า Q_f และ R_f นี้จะขึ้นอยู่กับแต่ละระบบ ซึ่งผู้ออกแบบระบบควบคุมของแต่ละระบบจะต้องเป็นผู้ออกแบบค่าเหล่านี้เอง

6.2 ข้อเสนอแนะของระบบควบคุมการเคลื่อนที่ของเครน

1.การวัดสัญญาณเอาท์พุทของระบบนั้น เราควรจะกรองเอาสัญญาณรบกวนออกจากสัญญาณเอาท์พุทก่อนที่จะนำสัญญาณนั้นไปใช้ในการควบคุมระบบ ซึ่งต้องรู้ความถี่ของสัญญาณรบกวนที่ปะปนเข้ามากับสัญญาณเอาท์พุทของระบบ

2.ในการออกแบบระบบควบคุมแบบ LQR และ LQG นั้นเราควรทดลองเปลี่ยนแปลงค่าพารามิเตอร์ Q , R และ Q_f , R_f ของระบบ LQR และ LQG ตามลำดับหลายๆค่าเพื่อที่จะหาค่าที่เหมาะสมที่สุดของระบบควบคุมของเรา

3.ควรเพิ่มเติมระบบชักรอกเพื่อให้สามารถเปลี่ยนแปลงความยาวของเชือกที่แขวนภาระอยู่ เพื่อให้ระบบที่ได้มีความใกล้เคียงกับระบบจริงและสามารถนำไปประยุกค์ใช้กับระบบจริงได้ง่ายขึ้น ซึ่งจะทำให้ระบบควบคุมการเคลื่อนที่ของเครนดังกล่าวเป็นระบบที่แปรค่าตามเวลา

ภาคผนวก ก

ทฤษฎีความน่าจะเป็น

1.ค่าเฉลี่ยและค่าความแปรปรวน(Mean and Variance)

สมมติ random vector $Z \in R^n$,โดยให้ $f_z(\zeta)$ เป็น ฟังชั่นความหนาแน่นของความน่าจะเป็น (The Probability Density Function : PDF) PDF แสดงถึงความน่าจะเป็นของ Z ที่ตำแหน่งต่างๆของค่า $d\zeta$ โดยมีจุดกึ่งกลางอยู่ที่ ζ ซึ่งบางทีเราอาจไม่รู้ค่าที่แน่นอนของ Z โดยเราอาจทราบเพียงว่า Z ที่เท่าใดมีความน่าจะเป็นมากที่สุดจาก PDF

ค่าเฉลี่ยหรือค่าความคาดหมาย(Expected value) ของฟังชัน g(z) ของ random vector Z เขียนได้ดังนี้

$$E\{g(z)\} = \int_{-\infty}^{\infty} g(\zeta) f_z(\zeta) d(\zeta) \qquad \text{----------(ก.1)}$$

ดังนั้นค่าเฉลี่ย(mean)หรือค่าความคาดหมาย(Expected value) ของ Z เขียนได้ดังนี้

$$E(z) = \int_{-\infty}^{\infty} \zeta f(\zeta) d\zeta \qquad \text{----------(ก.2)}$$

สามารถเขียนแทนค่าเฉลยของ Z ได้ด้วย $\bar{Z}$, $Z \in R^n$

ขบวนการหาค่าเฉลี่ยนั้นมีคุณสมบัติเชิงเส้น เพราะฉะนั้นเมื่อกำหนดตัวแปร random x,y และ ตัวแปรสเกลล่า a,b จะได้ว่า

$$\overline{ax + by} = a\bar{x} + b\bar{y} \qquad \text{----------(ก.3)}$$

ความแปรปรวน(Covariance)ของ $Z \in R^n$ แสดงได้โดย

$$P_z = E\{(z - \bar{z})(z - \bar{z})^T\} \qquad \text{----------(ก.4)}$$

โดยที่จะได้ว่า P_z เป็น n*n Positive definition matrix ใช้คุณสมบัติความเป็นเชิงเส้นสามารถเขียนสมการที่ (ก.4) ใหม่ได้ดังนี้

$$P_z = E\{zz^T\} - \bar{z}z^T \qquad \text{----------(ก.5)}$$

ซึ่งเราเรียก $E\{zz^T\}$ ว่า mean-square value ของ Z

ลักษณะ Random variable ที่สำคัญคือ Gaussian หรือ normal PDF ซึ่งมีการกระจายของค่าความน่าจะเป็น เป็นแบบเส้นโค้งปกติ

$$f_z(\zeta) = \frac{1}{\sqrt{(2\pi)^n |P_z|}} e^{-1/2(\zeta - \bar{z})^T P_z^{-1} (\zeta - \bar{z})} \qquad \text{----------(ก.6)}$$

ในกรณีที่ $Z \in R^n$ เป็นสเกลล่า n จะเท่ากับ 1

$$f_z(\zeta) = \frac{1}{\sqrt{2\pi P_z}} e^{-(\zeta - \bar{z})^2 / 2P_z} \qquad \text{----------(ก.7)}$$

ค่าความน่าจะเป็นของ random vector จะมีค่ามากขึ้นถ้า random vector มีค่าเข้าใกล้ค่าเฉลี่ยหรือค่าความคาดหมาย($\bar{Z}$) มากขึ้น

2.ความสัมพันธ์ระหว่าง random variable

สมมติ random vector $Z \in R^n$, $X \in R^m$ ให้ฟังชั่นความหนาแน่นของความน่าจะเป็นร่วม(joint probability density) ของ Z,X แสดงด้วย $f_{Z,X}(\zeta,\xi)$

ค่าความคาดหมาย (Expected value) ของฟังชั่น g(z,x) ของ random vectors Z และ X สามารถเขียนได้ดังนี้

$$E\{g(z,x)\} = \int_{-\infty}^{\infty} g(\zeta,\xi) f_{Z,X}(\zeta,\xi) d\zeta d\xi \qquad \text{----------(ก.8)}$$

ค่า Cross-Covariance แสดงถึงการขึ้นต่อกันของสอง random variable $Z \in R^n$, $X \in R^m$ สามารถเขียนได้ดังนี้

$$P_{z,x} = E\{(z-\bar{z})(x-\bar{x})^T\} \qquad \text{----------(ก.9)}$$

P_{zx} เป็น n*m matrix และ เราสามารถบอกได้ว่า random variable สองตัวไม่ขึ้นต่อกันเมื่อ

$$f_{Z,X}(\zeta,\xi) = f_Z(\zeta) f_X(\xi) \qquad \text{----------(ก.10)}$$

ภาคผนวก ข

การใช้งานการ์ดอินเตอร์เฟส PCL-714

1. คุณสมบัติทั่วไปของการ์ด PCL-714

- ขนาด 34 cm X 9.5 cm
- ใช้บัสของ IBM PC Bus
- ใช้สล็อต(Slot) แบบ 62 pin 1 สล็อต
- เลือกใช้ I/O Port Base Address ได้ในช่วง hex 200-hex 3F0

ข้อควรระวัง สำหรับสัญญาณอินพุตอนาลอกไม่ควรสูงกว่า +12.5 V และไม่ควรต่ำกว่า -5.5 V เพราะอาจทำให้การ์ดเสียหายได้

2. คุณสมบัติของวงจรแปลงสัญญาณอนาลอกเป็นดิจิตอลบนการ์ด PCL-714

- A/D มีความละเอียด 14 บิต จำนวน 16 ช่องสัญญาณแบบ differential
- A/D มี conversion time < 40 μsec
- มีย่านสัญญาณอินพุต -5 V ถึง +5 V
- มีอินพุตอิมพีแดนซ์ > 10 MΩ

3. การเลือก Base Address

เราสามารถเลือก Base Address ที่จะใช้ในการติดต่อกับ PCL-714 ได้โดยการเปิดปิดสวิทช์ 5 ตำแหน่งบนดิปสวิทช์ (SW1,ดูตำแหน่งของ SW1 ในรูป ข.1) ดังนี้

I/O Port Address (HEX)	Switch Position				
	1	2	3	4	5
	A8	A7	A6	A5	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

I/O Port Address (HEX)	Switch Position				
	1	2	3	4	5
	A8	A7	A6	A5	A4
3E0-3EF	1	1	1	1	0
3F0-3FF	1	1	1	1	1

ตาราง ข.1 ตารางแสดงการเลือก Base Address ของการ์ด PCL-714

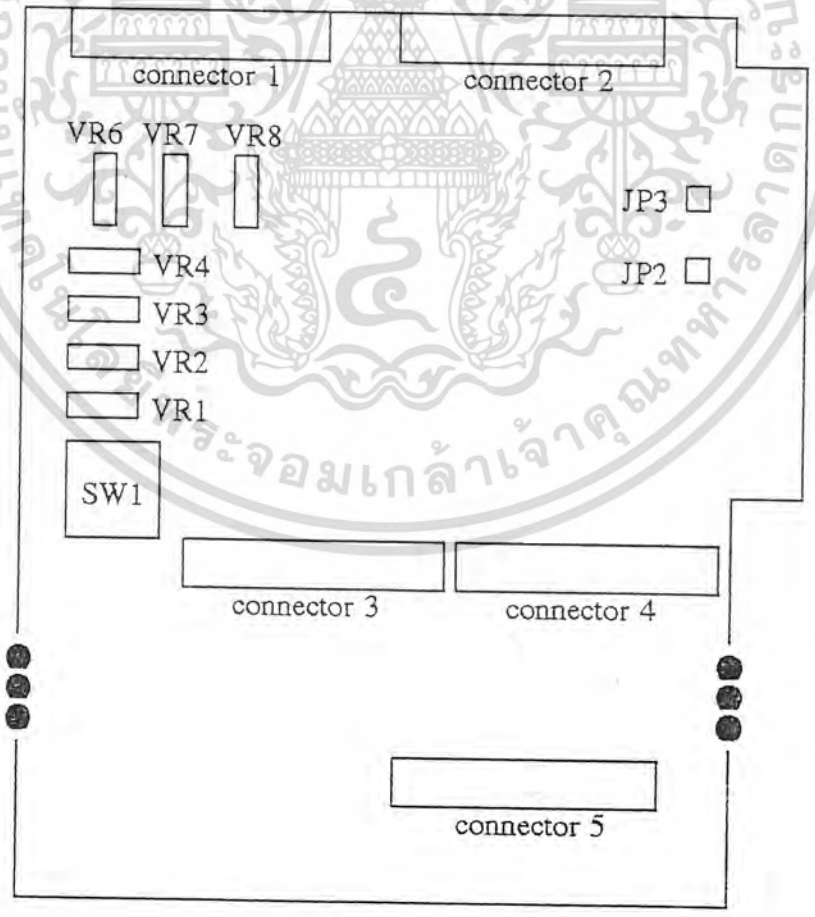
เมื่อ 0 หมายถึง ON

1 หมายถึง OFF

1...5 หมายถึง ตำแหน่งของสวิทช์

A4...A8 หมายถึง ตำแหน่งของแอดแดรสจาก IBM PC Bus

* หมายถึง เป็นตำแหน่งของสวิทช์ที่ตั้งค่ามาจากโรงงาน



รูปที่ ข.1 รูปแสดงส่วนประกอบโดยคร่าวๆของการ์ด PCL-714

4. ขาสัญญาณต่างๆของคอนเนคเตอร์

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

Connector 1 Connector 2

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

Connector 3 Connector 4

	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
+5 V	19	20	

Connector 5

เมื่อ 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 - clock input for 8253

GATE - gate input for 8253

OUT - signal output of 8253

5. โครงสร้างและรูปแบบของรีจิสเตอร์

ในหัวข้อนี้จะแสดงถึงหน้าที่ของรีจิสเตอร์และความสัมพันธ์ของกับรีจิสเตอร์แต่ละตัวกับ Base Address

ให้ LSB - Least Significant Byte

MSB - Most Significant Byte

R - Read operation on that byte

W - Write operation on that byte

BASE+0	R	LSB หรือ MSB ของ counter 0
	W	LSB หรือ MSB ของ counter 0
BASE+1	R	LSB หรือ MSB ของ counter 1
	W	LSB หรือ MSB ของ counter 1
BASE+2	R	LSB หรือ MSB ของ counter 2
	W	LSB หรือ MSB ของ counter 2
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) ถ้า บิตที่ 6 = 1 ,หมายถึง การแปลงข้อมูลของ A/D ยังไม่เสร็จ ถ้า บิตที่ 6 = 0 ,หมายถึง การแปลงข้อมูลของ A/D เสร็จแล้ว
	W	CH1 D/A high byte
BASE+6	R	D/I chanel 0-7
	W	CH2 D/A low byte

BASE+7	R	D/I channel 8-45
	W	CH2 high byte (bit 0-5)
BASE+8		Not used
BASE+9		Not used
BASE+10	W	Bit 0-3 ถูกใช้เพื่อเลือกช่องสัญญาณที่ต้องการติดต่อกับ A/D
BASE+11	W	ใช้ควบคุมโหมดการ trigger ของ A/D
BASE+12		Not used
BASE+13	W	D/O channel 0-7
BASE+14	W	D/O channel 8-15

6. วงจรแปลงสัญญาณอนาลอกเป็นดิจิตอล (Analog to Digital Conversion,A/D)

6.1 รูปแบบของข้อมูล

วงจร A/D นี้ใช้งานรีจิสเตอร์ 4 ตัวดังนี้

BASE+4 : เป็นข้อมูล byte ต่ำ(8 บิต)ของข้อมูลดิจิตอลจากทั้งหมด 14 บิตที่ได้จากการแปลงข้อมูลอนาลอก

BASE+5 : บิตที่ 0-5 เป็นข้อมูลส่วนที่เหลือ(byte สูง)ของข้อมูลดิจิตอลที่ได้จากการแปลงข้อมูล ส่วนบิตที่ 6 เป็นบิตสถานะของการแปลงข้อมูล ถ้าบิตนี้เป็น 0 หมายถึงการแปลงข้อมูลเสร็จแล้ว แต่ถ้าบิตนี้เป็น 1 หมายถึงการแปลงข้อมูลยังไม่เสร็จ

BASE+10 : บิตที่ 0-3 ของรีจิสเตอร์นี้ใช้เลือกช่องสัญญาณของ A/D (มีทั้งหมด 16 ช่องสัญญาณ) ที่ต้องการติดต่อด้วย ส่วนบิตที่ 4-7 ไม่ได้ใช้งาน

BASE+11 : เป็นรีจิสเตอร์ที่ใช้ควบคุม Trigger Mode(จะกล่าวรายละเอียดในหัวข้อถัดไป) ซึ่งมี 2 โหมด ได้แก่ Direct Trigger Mode และ Pacer Trigger Mode

บิตที่ 0 ถ้าเป็น 0 หมายถึง ไม่ใช้งานโหมด Direct Trigger

ถ้าเป็น 1 หมายถึง ใช้งานในโหมด Direct Trigger

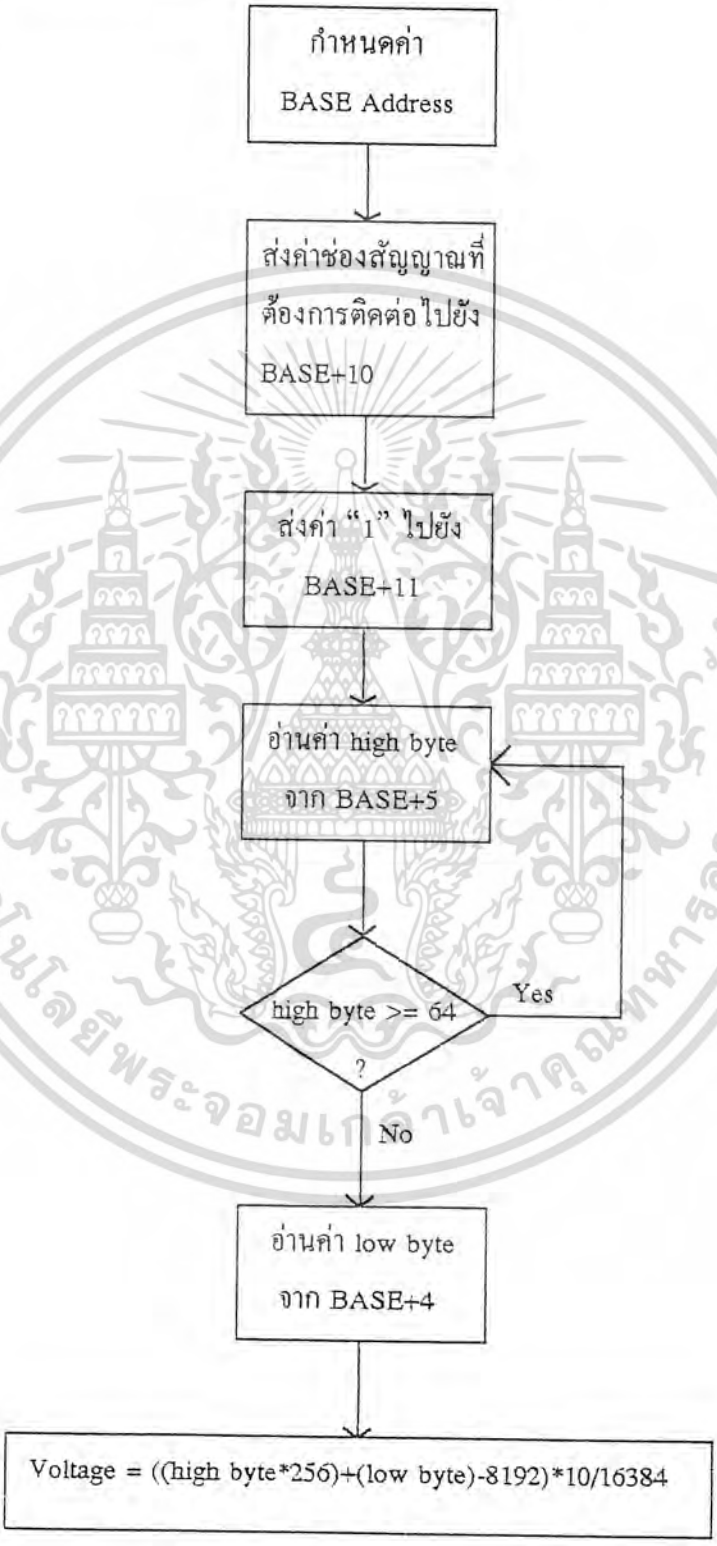
บิตที่ 1 ถ้าเป็น 0 หมายถึง ไม่ใช้งานโหมด Pacer Trigger

ถ้าเป็น 1 หมายถึง ใช้งานในโหมด Pacer Trigger

6.2 Trigger Mode

เราจะสามารถอ่านค่าจาก A/D เข้ามาได้เมื่อการแปลงข้อมูลของ A/D นั้นเสร็จแล้ว(บิตที่ 6 ของ BASE+5 เป็น 0) และทุกครั้งที่ต้องการอ่านค่าจาก A/D จะต้องทำการเลือกช่องสัญญาณที่ต้องการติดต่อและต้องทำการส่งสัญญาณ trig ไปให้ A/D ก่อน ซึ่งมีโหมดในการส่งสัญญาณการ trig อยู่ 2 โหมดดังนี้

1.Direct Trigger Mode เป็นการส่งสัญญาณการ trig โดยตรง หมายถึงทุกครั้งที่ต้องการอ่านค่าจาก A/D ต้องทำการส่งข้อมูล "1" มายัง BASE+11 นั่นเอง (ดูตัวอย่างโปรแกรมในส่วน source code) ซึ่งก็เหมือนกับการส่งสัญญาณ trig มาโดยตรงนั่นเอง



รูปที่ ข.2 โฟลชาร์ทการทำงานในแบบ Direct Trigger Mode

2. Pacer Trigger Mode เป็นโหมดการทำงานที่ใช้ 8253 timmer/counter ในการส่งสัญญาณ trig โดยสามารถกำหนดช่วงหรือคาบเวลาการส่งพัลส์สัญญาณ trig ได้ ซึ่งเราสามารถกำหนดคาบเวลาได้จากการให้ค่าการนับกับ 8253 timer/counter ซึ่งมีตัวอย่างการกำหนดค่าคาบเวลาที่ต้องการดังนี้

```
outport (BASE+3,#H74) ; ส่งค่า control byte ไปยัง 8253
outport (BASE+1,LB1) ;
outport (BASE+1,HB1) ;
outport (BASE+3,#HB4); ส่งค่า control byte ไปยัง 8253
outport (BASE+2,LB2) ;
outport (BASE+2,HB2) ;
```

โดย ค่าคาบเวลาการส่งพัลส์หรือคาบเวลาในการอ่านค่า(Sampling Period) (microseconds) ดังนี้

Sampling Period = (HB1*256 + LB1)*(HB2*256 +LB2)*0.5 microseconds

ส่วนการอ่านค่าก็เหมือนกับแบบ Direct Trigger คือต้องเลือกช่องสัญญาณที่ต้องการติดต่อก่อนทุกครั้งแต่แทนที่จะตามด้วยการส่งค่า "1" ไปยัง BASE+11 ก็ทำการส่งค่า "2" ไปยัง BASE+11 แทนแล้วรอจนครบคาบเวลาที่ตั้งไว้ จึงจะสามารถอ่านค่าได้ เพราะใน Pacer Mode ต้อง set ให้ค่าบิตที่ 1 ของ BASE+11 ให้เป็น 1 และ clear ให้บิตที่ 0 ของ BASE+11 เป็น 0 ซึ่งก็คือค่า "2" ในเลขฐานสิบนั่นเอง

ใน Pacer Mode เราสามารถ disable การทำงานโหมดนี้ โดยการให้สัญญาณ low กับ ขา GATE1 ที่ connector 4

7. รูปแบบการติดต่อ 8253 Interval Timer

8253 เป็น timer/counter ที่ประกอบด้วย counter 16 บิต 3 ตัวที่แยกจากกัน นับความถี่ได้สูงสุด 2.6 MHz counter แต่ละตัวสามารถโปรแกรมให้เริ่มนับได้ตั้งแต่ 0 ถึง 65,535 และมีโหมดการทำงานให้เลือก 6 โหมด (ดูรายละเอียดใน data sheet ของ 8253) ซึ่งเราใช้โหมด 2 ในการใช้งาน

8253 ใช้รีจิเตอร์ 4 ตัวในการติดต่อดังนี้

BASE+0 : อ่านและเขียนข้อมูล กับ counter 0

BASE+1 : อ่านและเขียนข้อมูล กับ counter 1

BASE+2 : อ่านและเขียนข้อมูล กับ counter 2

BASE+3 : เขียน Control Byte

ก่อนที่จะทำการเขียนหรืออ่านข้อมูลใดๆกับ 8253 จะต้องทำการส่ง Control Byte ที่บอกรายละเอียดในการติดต่อ ไปให้ 8253 ก่อน ซึ่งแต่ละบิตของ Control Byte มีความหมายดังนี้

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

1. SC1 และ SC0 ใช้เลือก counter

SC1	SC0	Counter
0	0	0
0	1	1
1	0	2
1	1	ไม่ใช้งาน

2. RL1 และ RL0 ใช้เลือกรูปแบบการอ่านเขียน ข้อมูล

RL1	RL0	Operation
0	0	Counter latch
0	1	อ่านเขียน LSB
1	0	อ่านเขียน MSB
1	1	อ่านเขียน LSB ก่อนแล้วตามด้วย MSB

3. M2 ,M1 และ M0 ใช้เลือกโหมดการทำงาน

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

4. BCD ใช้เลือกรูปแบบชนิดของข้อมูลการนับ

BCD	ชนิดข้อมูล
0	Binary Counter 16 bits
1	BCD Counter

ข้อมูลละเอียดของการใช้งานส่วนอื่นดูได้ในคู่มือการใช้งานการ์ด PCL-714

(ดูตัวอย่างการโปรแกรมที่ใช้งานทั้งหมดได้ใน source code ใน header file ชื่อ ADC.H)

00714-90001 March, 1988 Rev. 2B

Table of Contents

1. INTRODUCTION TO THE PCL-714

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).

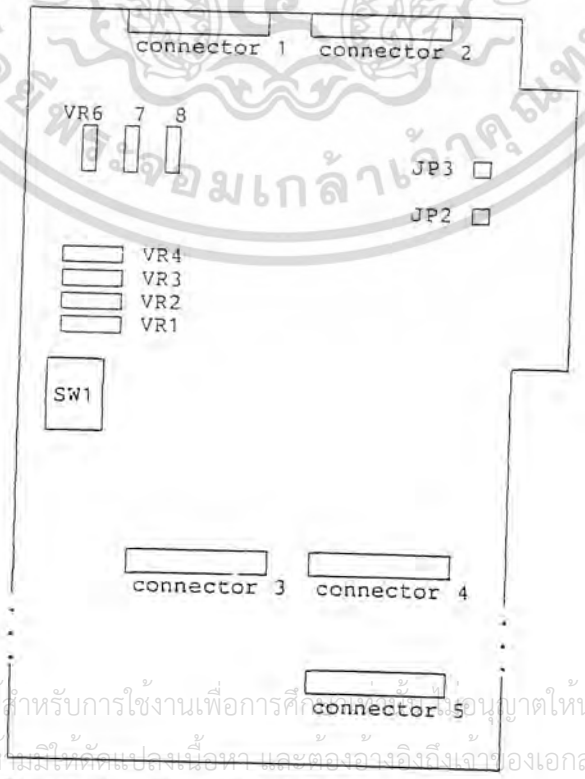


Figure 2.1 The PCL-714 Card

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.

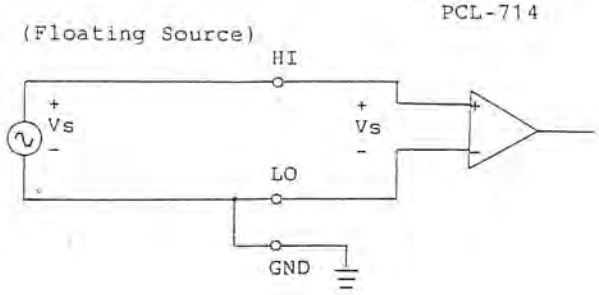
Secure the PCL-714 with the 3/16" mounting screw, then attach an appropriate cable to the connector.

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

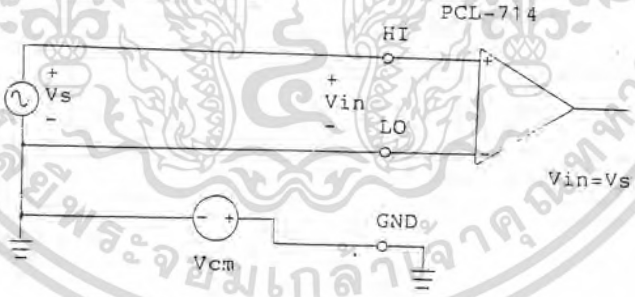
2.6.1. Analog Signal Connection

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:

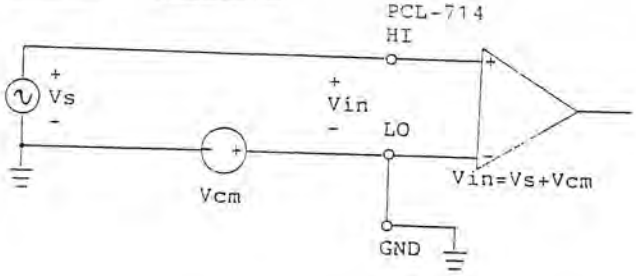


A connection must exist between LO and GND to define Common input voltage.
If the signal source has one side connected to a local ground. The signal source ground and the PCL-714 ground will not be at the same voltage as they are connected through the ground return of the equipment and building wiring. 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

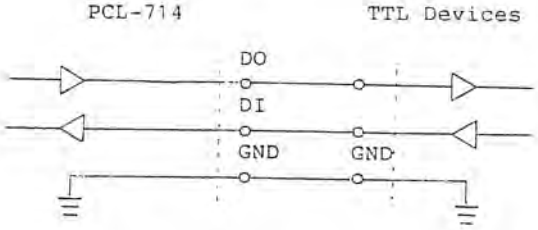


Incorrect Connection

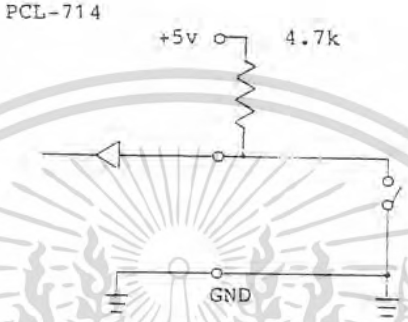


2.6.2. Digital Signal Connection

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 high 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. Ch.1,2 used as pacer
	W	LSB or MSB of Counter 0	
BASE+1	R	LSB or MSB of Counter 1	
	W	LSB or MSB of Counter 1	
BASE+2	R	LSB or MSB of Counter 2	
	W	LSB or MSB of Counter 2	
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) bit 6 =1, A/D conversion not ready 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	
	W	CH2 D/A high byte (bit 0-5)	

```
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-711 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 very flexible. Only two conventions need to be remembered:

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

Appendix B: Calibration

The PCL-714 is equipped with 7 variable resistors (VR1-VR4, VR6-VR8) to calibrate the A/D and D/A converters. The functions of the variable resistors are :

NAME	ADJUSTMENT
VR1	Offset of D/A ch. 1
VR2	Offset of D/A ch. 2
VR3	Offset of A/D conversion
VR4	Common mode rejection of A/D
VR5	Reserved
VR6	Full scale of D/A ch. 1
VR7	Full scale of D/A ch. 2
VR8	Full scale of A/D conversion

To perform a D/A calibration, set both D/A channels to 0 volt by setting the corresponding registers with the value 8192. Use a calibrated digital voltmeter to measure the D/A channel 1 (2). Adjust VR1 (2) to make D/A channel 1 (2) have 0 volt outputs. Then, set both D/A channels to 5V by setting the corresponding registers with the value 16384. Adjust VR6 (7) to make D/A channel 1 (2) have 5V outputs.

To perform a A/D calibration, use the following BASIC program to do the calibration. When the program is running, a value will be shown on the screen. Connect GND to both HI and LO input of channel 15. Adjust VR3 until the reading is 8192. This is the zero offset calibration. Apply standard +5V voltage to both HI and LO of channel 15 and adjust VR4 to get reading of 8192. This is common mode rejection calibration. Apply standard +5V to HI and GND to LO of channel 15 and adjust VR8 to get reading of 16383. This is full scale calibration.

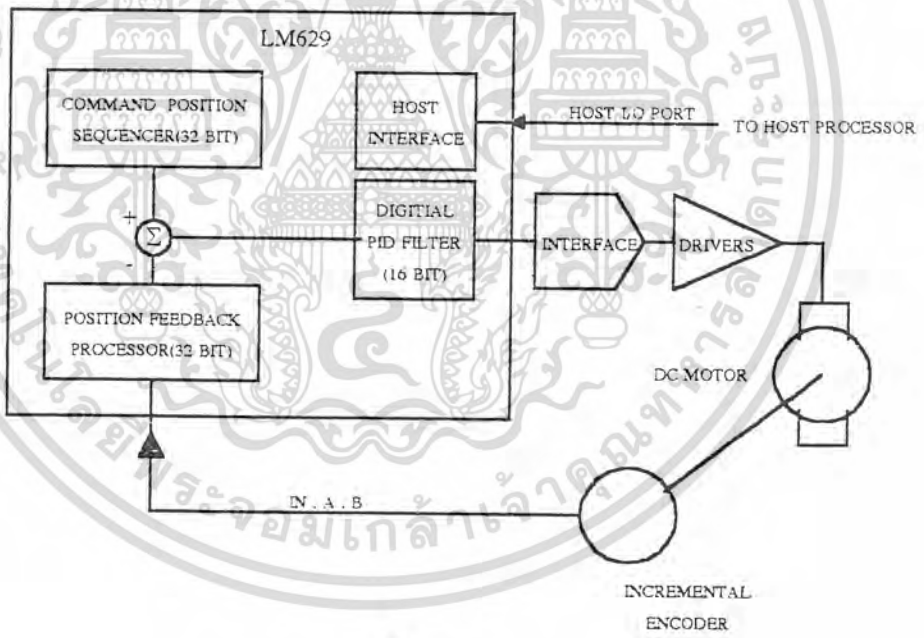
```
10  BASE%=&H220          'BASE address
20  CLS
30  OUT BASE%+10,15      'Select channel 15
40  SUM=0
50  FOR I=1 TO 10
60   OUT BASE%+11,1      'Trigger the A/D
70   'Loop if the conversion is not complete
80   HI=INP(BASE%+5) : IF HI>=64 THEN 80
90   LO=INP(BASE%+4)     'Read the low byte
100  V=HI*256+LO
110  SUM=SUM+V
120 NEXT I
130 LOCATE 10,10
140 PRINT SUM/10         'Print the reading
150 GOTO 40
160 END
```

ภาคผนวก ค

การใช้งาน LM629 และบอร์ดควบคุมความเร็ว

1 การใช้งานLM629

LM628/LM629 เป็นวงจรรวมที่ใช้ควบคุมการเคลื่อนที่ของมอเตอร์โดยรับสัญญาณป้อนกลับจากเอนโคดเดอร์ โดย Block Diagram การทำงานแสดงไว้ดังรูปที่ ค.1 , LM628 และLM629 นั้นมีโครงสร้างภายในเหมือนกันต่างกันตรงที่เอาท์พุทของLM628นั้นเป็นสัญญาณ Digital 8 หรือ 12 บิต แต่ LM629 ให้เอาท์พุทเป็นสัญญาน PWM ซึ่งประกอบด้วยสัญญานขนาด(Magnitude:เป็นความต่างศักย์ที่ป้อนให้มอเตอร์) และสัญญาณทิศทาง(sign:ทิศทางการหมุนของมอเตอร์) ซึ่งในปริญญานิพนนี้เราใช้ LM629 เพื่อให้ง่ายแก่การออกแบบวงจรขับกำลัง(Driver)



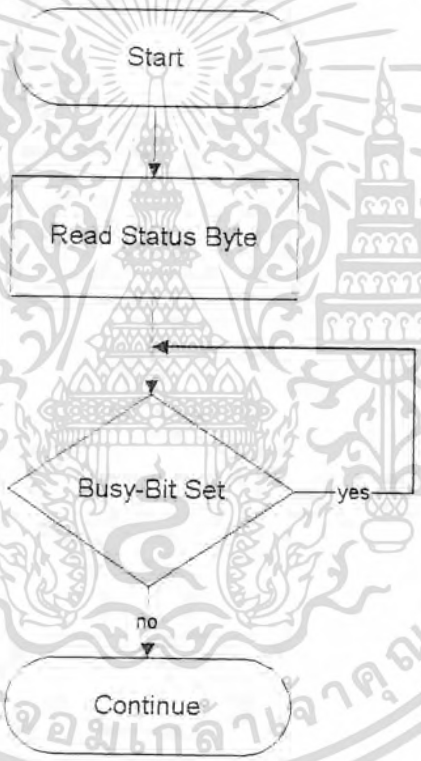
รูปที่ ค.1

การใช้งาน LM629 นั้นสามารถแบ่งเป็นขั้นตอนได้ดังนี้

1.1 ขบวนการตรวจสอบสภาวะพร้อมรับคำสั่ง(Busy-bit Check Module)

Busy-bit เป็นบิทแสดงสภาวะซึ่งจะเซตเป็น '1' ทุกครั้งหลังจาก Host(ในที่นี้คือคอมพิวเตอร์) เขียนไบท์คำสั่งหรือเขียน,อ่านไบท์ที่สองของคาต้าไปยัง ในระหว่างที่ Busy-bit อยู่ในสภาวะ '1' อยู่นั้น LM629 จะไม่สนใจคำสั่งหรือข้อมูลใดๆที่ส่งไปยังตัวมัน

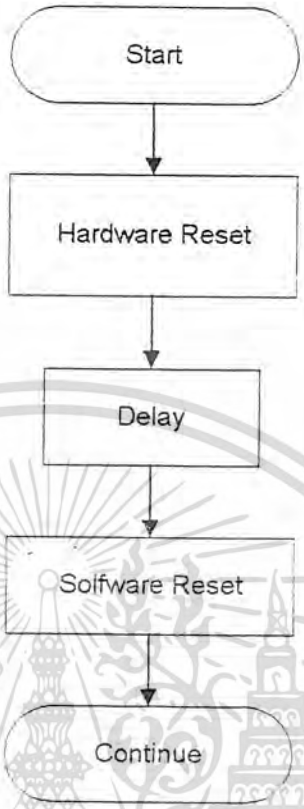
ฟังชันในการตรวจสอบสภาวะนี้คือ void WAITBUSY() ซึ่งอยู่ใน header file ชื่อ LM629V.H ซึ่งมีการทำงานดังรูปที่ ค.2



รูปที่ ค.2

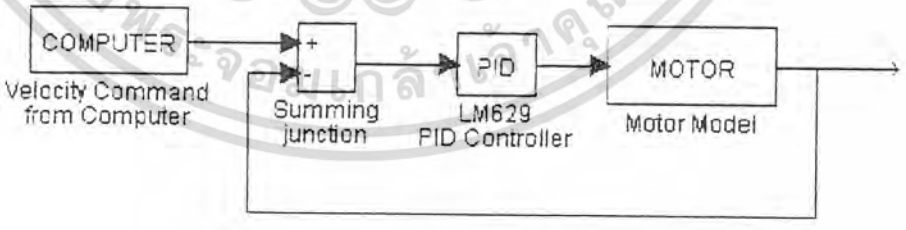
1.2ขบวนการเริ่มต้นการทำงาน(INTITIALIZATION)

การเริ่มต้นการทำงานของ LM629 ประกอบด้วยสองขั้นตอนคือ เริ่มต้นด้วยการทำ HARDWARE RESET คือ การให้ลอจิก '0' แก่ขา RESET ของLM629 (สามารถดู timing ได้จาก data sheet ท้ายภาคผนวก) ต่อจากนั้นหน่วงเวลาประมาณ 500 ms จากนั้นทำ SOLFWARE RESET โดยส่งคำสั่ง RESET ไปยัง LM629 รูปแสดงการทำงานของโมดูล INTITIALLIZATION แสดงดังรูปที่ ค.3 โดยโมดูลนี้คือ void INIT_LM629(void) ซึ่งอยู่ใน Header File ชื่อ LM629V.H



รูปที่ ค.3

1.3การออกแบบ PID ฟิลเตอร์ของลูปความเร็ว



รูปที่ ค.4

y : ความเร็วของตัวรถเครน(m/s)

r : คำสั่งความเร็วจากคอมพิวเตอร์(m/s)

จากบทที่ 4 เราได้ทำการคำนวนค่าฟิลเตอร์ไว้แล้ว ซึ่งจากฟิลเตอร์ที่ได้นั้นจะต้องนำมาสเกลก่อนที่จะป้อนให้กับ LM629 ดังจะกล่าวต่อในหัวข้อถัดไป

1.4 การสเกลค่าฟิลเตอร์พารามิเตอร์

เนื่องจากลักษณะโครงสร้างภายในของ LM629 นั้นแตกต่างจากโครงสร้างทางทฤษฎี เพราะฉะนั้นเราจึงต้องทำการสเกลค่าพารามิเตอร์ที่คำนวณได้ ก่อนที่จะป้อนให้LM629 ดังนั้น ก่อนอื่นเราจะมาพิจารณาโครงสร้างภายในของ LM629 ซึ่งเป็น PID Controller สามารถเขียนเอาท์พุทที่ออกจากคอนโทรนเลอร์ได้ดังนี้

สมการดีสครีทคอนโทรนเลอร์ของระบบที่ออกแบบไว้ในบทที่ 4 :

$$u(k) = K_p e(k) + K_i T\sum_{k=0}^{K} e(k) + K_d / T_s\left[e(k) - e(k-1)\right]$$

โดยที่

T : Sampling time = $2048/f_{CLK}$

T_s : Delivative Sampling time = $2048/f_{CLK}$*(1...255)

สมการดีสครีทคอนโทรนเลอร์ของ LM629:

$$u(k) = k_p e(k) + k_i \sum_{k=0}^{K} e(k) + k_d\left[e(k') - e(k'-0)\right]$$

โดยที่

k_p,k_i,k_d : ดีสครีทฟิลเตอร์พารามิเตอร์ของ LM629

e(k) : position error ที่เวลา k

k′ : การชักตัวอย่างที่ Derivative Sampling Rate

e(k) ซึ่งออกจาก Summing Junction มีขนาด 16 บิท ถูกนำไปคูณกับ ฟิลเตอร์พารามิเตอร์ขนาด 15 บิท ได้ผลลัพท์ขนาด 32 บิท โดย 16 บิทล่างจาก 32 บิท จะถูกนำมารวมกันเป็น u(k) ในทุกๆ ช่วงเวลาการชักตัวอย่าง โดยนำเอาเฉพาะ 8 บิทบนของ u(k) มาเป็นเอาท์พุท PWM ดังรูปที่ ค.5

รูปที่ ค.5

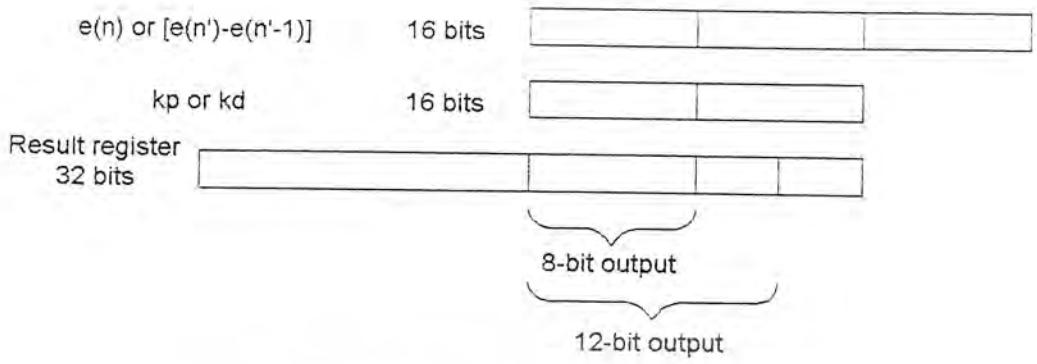
1.4.1 การสเกลค่า k_p และ k_i จาก รูปที่ ค.5 จะเห็นว่าได้นำเอาเฉพาะ 8 บิทบนของ u(k) มาเป็นเอาท์พุท ซึ่งก็คือการหารค่า k_p และ k_i ด้วย 256 สามารถคำนวนค่า k_p และ k_i ได้ดังนี้

$$k_p * e(k) / 256 = K_p * e(k)$$

$$k_p = 256K_p$$

$$\left(k_d * \left[e(k') - e(k'-1)\right]\right) / 256 = K_d / T_s * e(k)$$

$$k_d = 256K_d / T_s$$

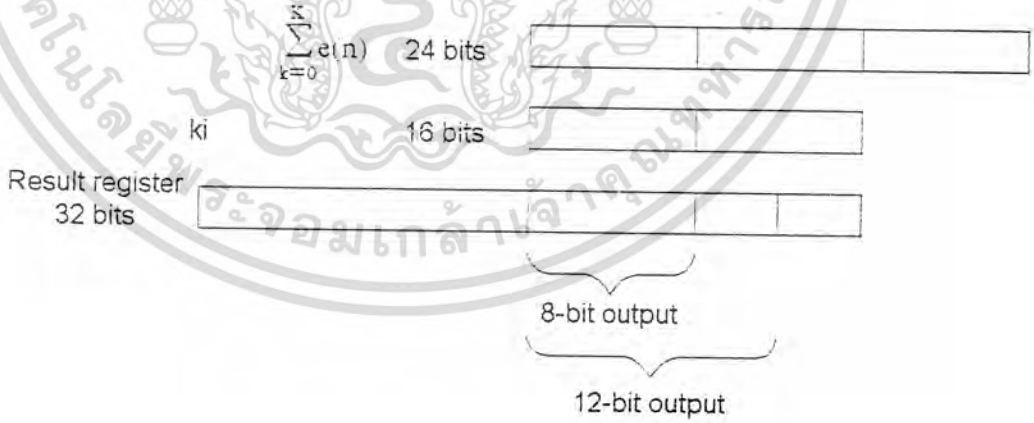


รูปที่ ค.5

1.4.2 การสเกลค่า k_i จากรูปที่ ค.6 ผลรวมของค่า error ในเทอม Integral มีขนาด 24 บิท ซึ่ง LM629 จะนำเฉพาะ 16 บิท บน มาคูณกับ k_i ได้ผลลัพท์ขนาด 32 บิท แล้วจึงนำเฉพาะ 16 บิท ล่างของแต่ละเทอมมารวมกันเป็น u(k) ต่อจากนั้นนำเฉพาะ 8 บิทบนมาเป็นเอาท์พุท PWM จะเห็นได้ว่ามีการเลื่อนไปทางขวา 8 บิท จำนวน 2 ครั้ง เขียนเป็นสมการได้ดังนี้

$$k_i / 256 * e(k) \quad 256 = K_i * T$$

$$k_i = 65536 * K_i * T$$



รูปที่ ค.6

1.5 การโปรแกรมค่าฟิลเตอร์พารามิเตอร์

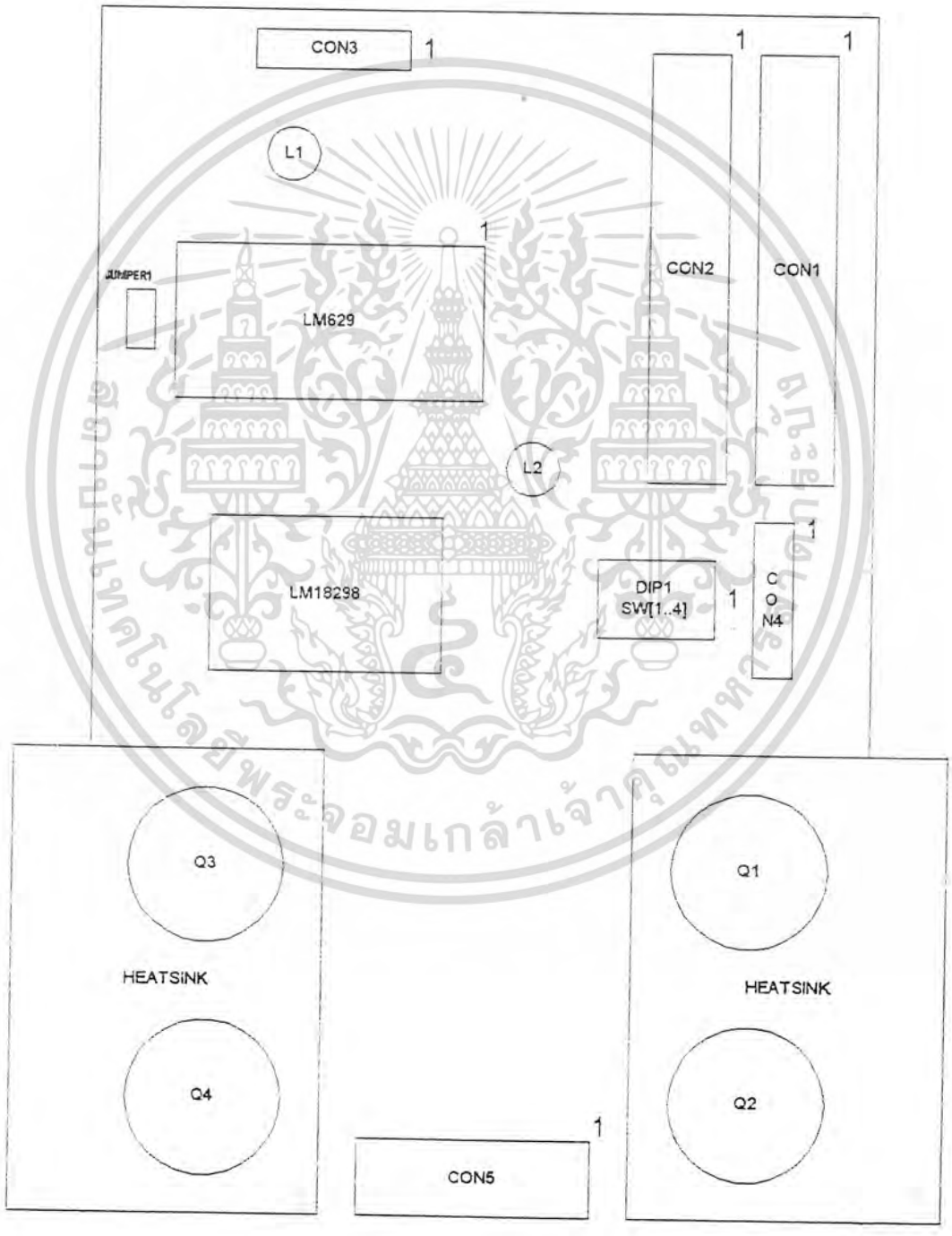
ในการโปรแกรมค่า PID ฟิลเตอร์พารามิเตอร์นั้นประกอบด้วยสองขั้นตอนคือ

1.การป้อนค่าฟิลเตอร์พารามิเตอร์โดยใช้คำสั่ง LFIL ซึ่ง LM629 จะนำค่านี้ไปเก็บเอาไว้ใน Buffer Register โดยค่าฟิลเตอร์นี้จะยังไม่มีผลต่อการทำงานของ LM629

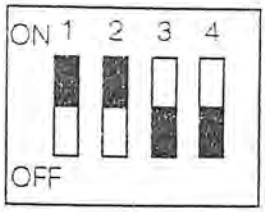
2.การอัพเดทฟิลเตอร์พารามิเตอร์โดยใช้คำสั่ง UDF หลังจากที่ป้อนค่าฟิลเตอร์ให้ LM629 แล้วค่าฟิลเตอร์นั้นยังอยู่ใน Buffer Register จนกระทั่งได้ส่งคำสั่ง UDF , ฟิลเตอร์นั้นจึงจะถูกนำไปใช้

2 ภาพแสดงรูปแผ่นวงจรคาบคุมความเร็วของมอเตอร์ผ่านLM629

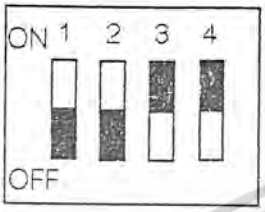
วงจรที่ใช้ในการควบตุมความเร็วผ่าน LM629 นั้นออกแบบตามวงจรในรูปที่ 2.9ก และ 2.9ข ในบทที่ 2 โดยมีตำแหน่งการวางอุปกรณ์ต่างๆดังนี้



รูปที่ ค.7



เมื่อเซต DIP1 SW[1..4] ดังรูป บอร์ดควบคุมความเร็วจะรับคำสั่งความเร็วจาก LM629



เมื่อเซต DIP1 SW[1..4] ดังรูป บอร์ดควบคุมความเร็วจะรับคำสั่งความเร็วจาก CON4

L1	แสดงสภาวะความพร้อมของแหล่งจ่าย 12 vที่ใช้กับเอนโคดเดอร์
L2	แสดงสภาวะความพร้อมของแหล่งจ่าย 5 v ที่ใช้ในวงจร
CON1,CON2	เป็นคอนเน็กเตอร์ที่ใช้เชื่อมต่อกับวงจรที่ดังรูปที่ 2.6 ที่คอนเน็กเตอร์ ETT_BUS
CON3	เป็นคอนเน็กเตอร์ที่ใช้เชื่อมต่อกับเอนโคดเดอร์
CON4	เป็นคอนเน็กเตอร์ที่ใช้เชื่อมต่อกับสัญญาณควบคุมจากภายนอก
CON5	เป็นคอนเน็กเตอร์ที่ใช้เชื่อมต่อกับมอเตอร์
JUMPER1	ใช้เลือกว่าจะให้มีการอินเทอร์รัพท์จาก LM629 หรือไม่

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 NMOS and packaged in a 28-pin dual in-line package or a 24-pin surface mount package (LM629 only). Both 6 MHz and 8 MHz maximum frequency versions are available with the suffixes -6 and -8, respectively, used to designate the versions. They incorporate an SDA core processor and cells designed by SDA.

Features

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

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

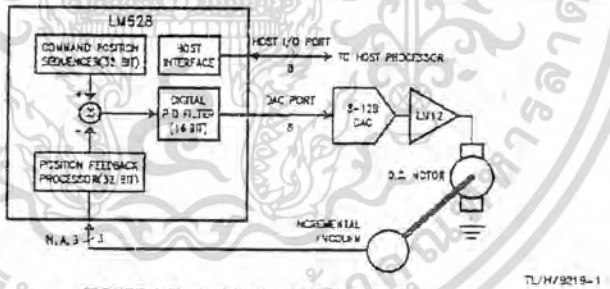


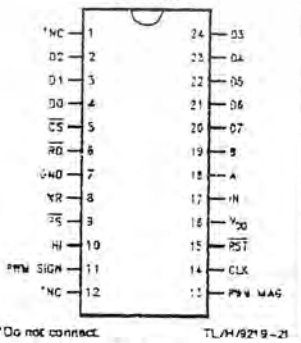
FIGURE 1. Typical System Block Diagram

Connection Diagrams

LM628N

LM629N

LM629M



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

RRD-B30M115/Printed in U. S. A.

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:	
LM628N-6, LM629N-6, LM629M-6	$1.0\ MHz < f_{CLK} < 6.0\ MHz$
LM628N-8, LM629N-8, LM629M-8	$1.0\ MHz < f_{CLK} < 8.0\ MHz$
V_{DD} Range	$4.5V < V_{DD} < 5.5V$

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

Symbol	Parameter	Conditions	Tested Limits		Units
			Min	Max	
I_{DD}	Supply Current	Outputs Open		110	mA
INPUT VOLTAGES					
V_{IH}	Logic 1 Input Voltage		2.0		V
V_{IL}	Logic 0 Input Voltage			0.8	V
I_{IN}	Input Currents	$0 < V_{IN} < V_{DD}$	−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	Tag	Tested Limits		Units
		Min	Max	
ENCODER AND INDEX TIMING (See *Figure 2*)				
Motor-Phase Pulse Width	T1	$\frac{16}{f_{CLK}}$		µs
Dwell-Time per State	T2	$\frac{8}{f_{CLK}}$		µs
Index Pulse Setup and Hold (Relative to A and B Low)	T3	0		µs
CLOCK AND RESET TIMING (See *Figure 3*)				
Clock Pulse Width				
LM628N-6, LM629N-6, LM629M-6	T4	78		ns
LM628N-8, LM629N-8, LM629M-8	T4	57		ns
Clock Period				
LM628N-6, LM629N-6, LM629M-6	T5	166		ns
LM628N-8, LM629N-8, LM629M-8	T5	125		ns
Reset Pulse Width	T6	$\frac{8}{f_{CLK}}$		µs

AC Electrical Characteristics (Continued)

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

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

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

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

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

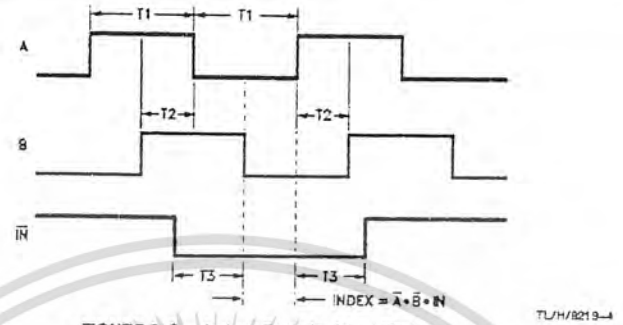


FIGURE 2. Quadrature Encoder Input Timing

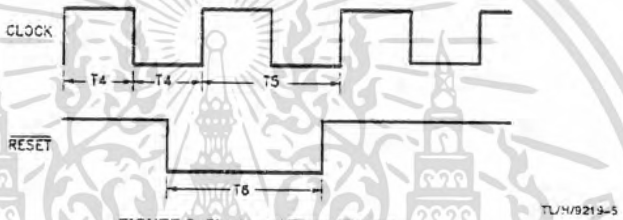


FIGURE 3. Clock and Reset Timing

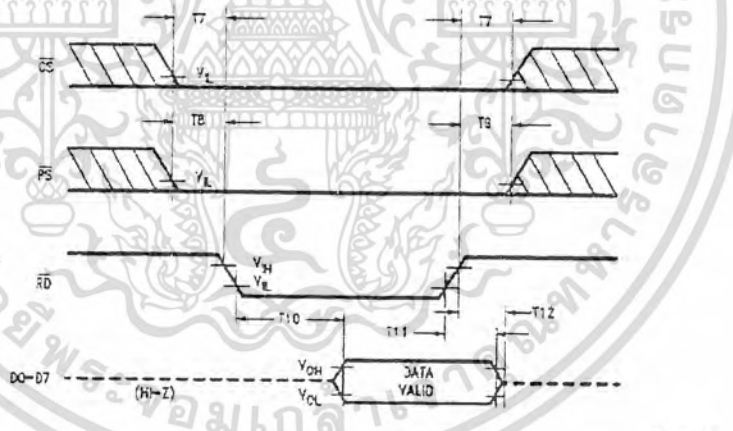


FIGURE 4. Status Byte Read Timing

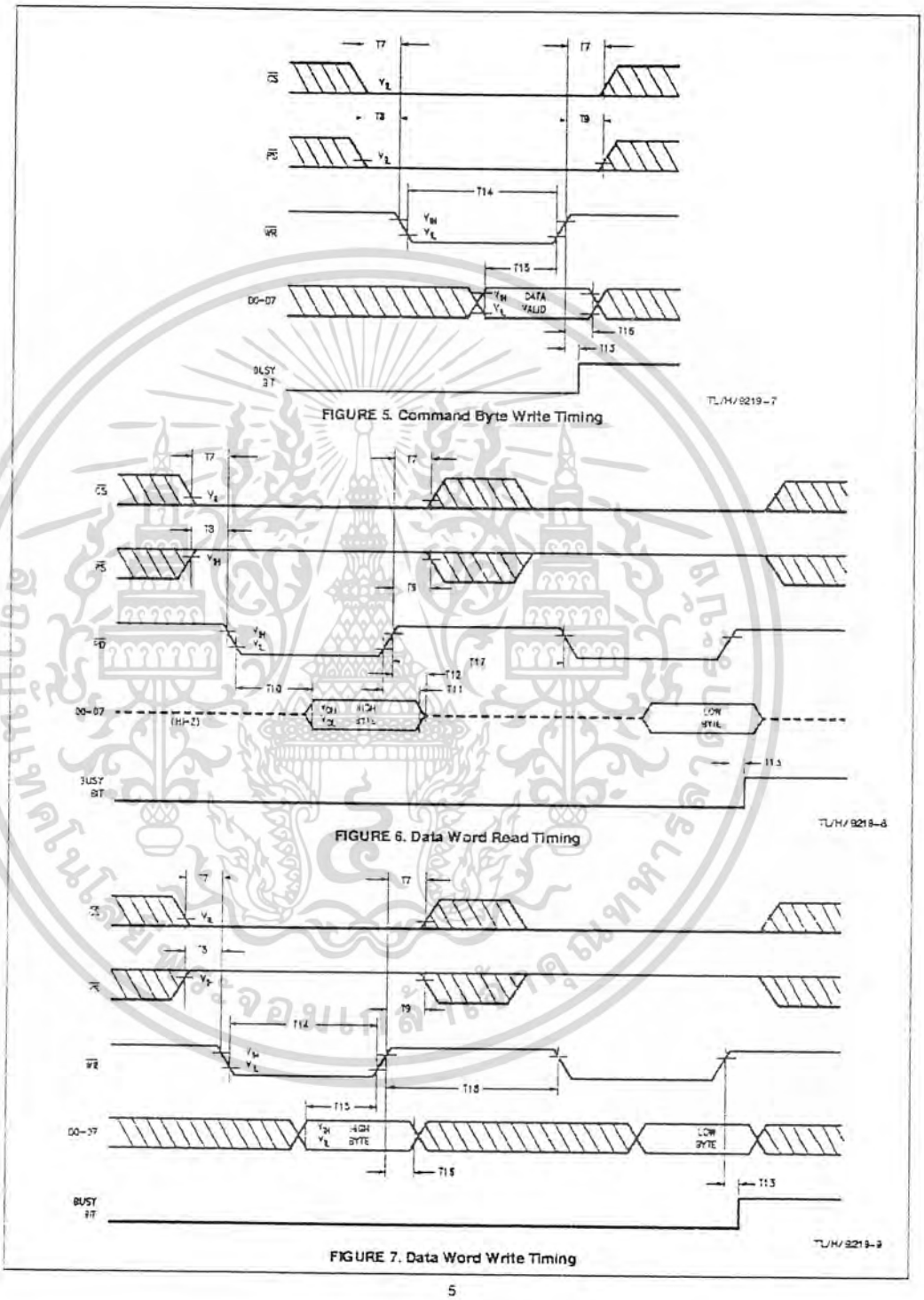


FIGURE 5. Command Byte Write Timing

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

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

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

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

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

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

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

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

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

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

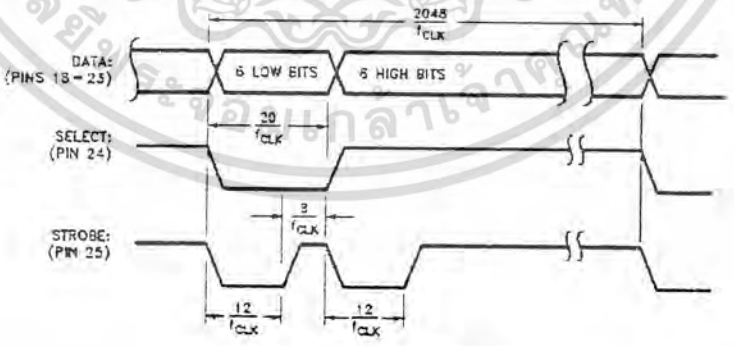
Pin 26 (14), Clock (CLK) Input: Receives system clock.

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

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

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

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



TL/H/9213–10

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 $1,073,741,823/2^{16}$ counts/sample; ie, 0 to 16,383 counts/sample, with a resolution of $1/2^{16}$ counts/sample
Acceleration Range	0 to $1,073,741,823/2^{16}$ counts/sample/sample; ie, 0 to 16,383 counts/sample/sample, with a resolution of $1/2^{16}$ counts/sample/sample
Motor Drive Output	LM628: 8-bit parallel output to DAC, or 12-bit multiplexed output to DAC LM629: 8-bit PWM sign/magnitude signals
Operating Modes	Position and Velocity
Feedback Device	Incremental Encoder (quadrature signals; support for index pulse)
Control Algorithm	Proportional Integral Derivative (PID) (plus programmable integration limit)
Sample Intervals	Derivative Term: Programmable from $2048/f_{CLK}$ to $(2048 * 256)/f_{CLK}$ in steps of $2048/f_{CLK}$ (256 to 65,536 μs for an 8.0 MHz clock). Proportional and Integral: $2048/f_{CLK}$

Theory of Operation (Continued)

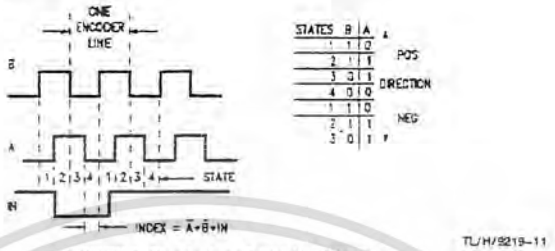


FIGURE 9. Quadrature Encoder Signals

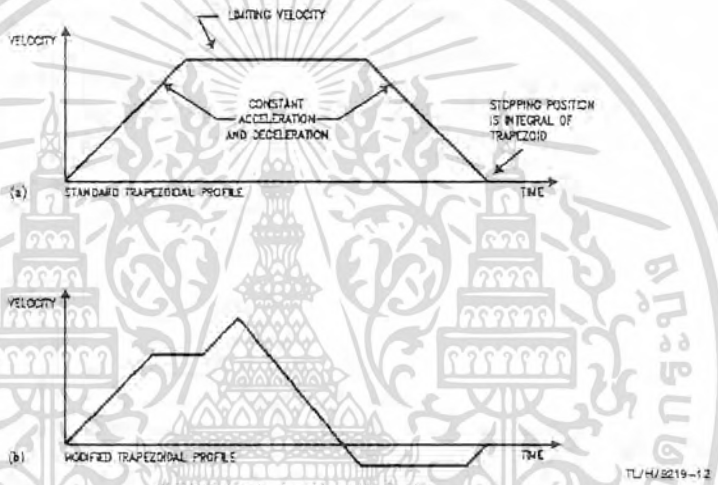


FIGURE 10. Typical Velocity Profiles

VELOCITY PROFILE (TRAJECTORY) GENERATION

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

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

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

Theory of Operation (Continued)

let P = target position (units = encoder counts)
let R = encoder lines * 4 (system resolution)
then R = 500 * 4 = 2000
and P = 2000 * desired number of revolutions
P = 2000 * 100 revs = 200,000 counts (value to load)
P (coding) = 00030D40 (hex code written to LM628)

let V = velocity (units = counts/sample)
let T = sample time (seconds) = 341 µs (with 6 MHz clock)
let C = conversion factor = 1 minute/60 seconds
then V = R * T * C * desired rpm
and V = 2000 * 341E−6 * 1/60 * 600 rpm
V = 6.82 counts/sample
V (scaled) = 6.82 * 65,536 = 446,955.52
V (rounded) = 446,956 (value to load)
V (coding) = 0006D1EC (hex code written to LM628)

let A = acceleration (units = counts/sample/sample)
A = R * T * T * desired acceleration (rev/sec/sec)
then A = 2000 * 341E−6 * 341E-6 * 1 rev/sec/sec
and A = 2.33E−4 counts/sample/sample
A (scaled) = 2.33E−4 * 65,536 = 15.24
A (rounded) = 15 (value to load)
A (coding) = 0000000F (hex code written to LM628)

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

PID COMPENSATION FILTER

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

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

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 this product is compared to coefficient il, and the lesser, appropriately signed magnitude then contributes to the motor control signal.

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

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

LM628 READING AND WRITING OPERATIONS

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

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

Writing and reading data to/from the LM628 (as opposed to writing commands and reading status) are done with $\overline{PS}$ (Pin 16) logic high. These writes and reads are always an integral number (from one to seven) of two-byte words, with the first byte of each word being the more significant. Each byte requires a write ($\overline{WR}$) or read ($\overline{RD}$) strobe. When transferring data words (byte-pairs), it is necessary to first read the status byte and check the state of the busy bit. When the

Theory of Operation (Continued)

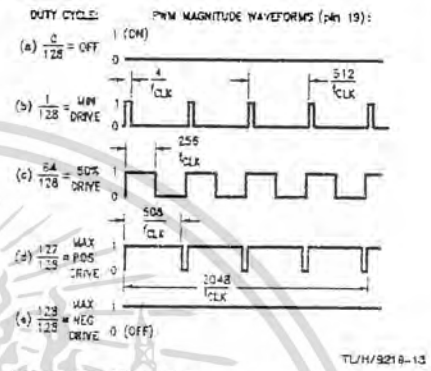
busy bit is logic low, the user may then sequentially transfer both bytes comprising a data word, but the busy bit must again be checked and found to be low before attempting to transfer the next byte pair (when transferring multiple words). Data transfers are accomplished via LM628-internal interrupts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



TL/H/9218–13

Note: Sign output (pin 18) not shown

FIGURE 11. PWM Output Signal Format

TABLE II. LM628 User Command Set

Command	Type	Description	Hex	Data Bytes	Note
RESET	Initialize	Reset LM628	00	0	1
PORT8	Initialize	Select 8-Bit Output	05	0	2
PORT12	Initialize	Select 12-Bit Output	06	0	2
DFH	Initialize	Define Home	02	0	1
SIP	Interrupt	Set Index Position	03	0	1
LPEI	Interrupt	Interrupt on Error	1B	2	1
LPES	Interrupt	Stop on Error	1A	2	1
SBPA	Interrupt	Set Breakpoint, Absolute	20	4	1
SBPR	Interrupt	Set Breakpoint, Relative	21	4	1
MSKI	Interrupt	Mask Interrupts	1C	2	1
RSTI	Interrupt	Reset Interrupts	1D	2	1
LFIL	Filter	Load Filter Parameters	1E	2 to 10	1
UDF	Filter	Update Filter	04	0	1
LTRJ	Trajectory	Load Trajectory	1F	2 to 14	1
STT	Trajectory	Start Motion	01	0	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 totally supported by hardware.

User Command Set

GENERAL

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

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

Initialization Commands

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

RESET COMMAND: RESET the LM628

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

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

PORT8 COMMAND: Set Output PORT Size to 8 Bits

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

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

PORT12 COMMAND: Set Output PORT Size to 12 Bits

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

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

DFH COMMAND: DeFine Home

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

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

Interrupt Control Commands

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

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

SIP COMMAND: Set Index Position

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

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

LPEI COMMAND: Load Position Error for Interrupt

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

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

Interrupt Control Commands (Continued)

LPES COMMAND: Load Position Error for Stopping

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

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

SBPA COMMAND:

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

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

SBPR COMMAND:

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

This command enables the user to set a breakpoint in terms of relative position. As with command SBPA, bit 6 of the status byte is set to logic high when the breakpoint position (relative to the current commanded target position) is reached. The relative breakpoint input value must be such that when this value is added to the target position the result remains within the absolute position range of the system (C0000000 to 3FFFFFFF hex). This condition is useful for signaling trajectory and/or filter parameter updates. The user can also arrange to have the LM629 interrupt the host to signify that a breakpoint position has been reached. See the descriptions for commands MSKI and RSTI.

MSKI COMMAND: MaSK Interrupts

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

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

TABLE III. Mask and Reset Bit Allocations for Interrupts

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

RSTI COMMAND: ReSeT Interrupts

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

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

Filter Control Commands

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

LFIL COMMAND: Load FILter Parameters

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

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

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

(see Eq. 1) to values equal to or less than a user-defined maximum value; this capability minimizes integral or reset "wind-up" (an overshooting effect of the integral action). The positive-only input value is compared to the absolute

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 step, 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 2048/8 MHz sample intervals must be scaled for other clock frequencies.

Trajectory Control Commands (Continued)

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

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

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

STT COMMAND: STarT Motion Control

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

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

Data Reporting Commands

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

Data Reporting Commands (Continued)

RDSTAT COMMAND: ReaD STATus Byte

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

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

TABLE VII. Status Byte Bit Allocation

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

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

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

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

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

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

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

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

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

RDSIGS COMMAND: ReaD SIGnalS Register

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

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

TABLE VIII. Signals Register Bit Allocation

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

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

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

Data Reporting Commands (Continued)

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

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

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

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

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

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

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

RDIP COMMAND: ReaD Index Position

Command Code:	09 Hex
Bytes Read:	Four
Data Range:	00000000 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:	00000000 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:	00000000 to 3FFFFFFF Hex
Executable During Motion:	Yes

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

RDDV COMMAND: ReaD Desired Velocity

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

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

RDRV COMMAND: ReaD Real Velocity

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

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

RDSUM COMMAND: ReaD Integration-Term SUMmation Value

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

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

Typical Applications

Programming LM628 Host Handshaking (Interrupts)

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

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 LM629 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 12 of the DAC socket (Pins 2 and 6 of the LF356) and adjust the 25 kΩ potentiometer for 0V at Pin 6 of the LF356.

If the DAC is not removable, the second method of adjustment requires that the DAC1210 inputs be presented an all-zeros code. This can be arranged by commanding the appropriate move via the LM628, but with no feedback from the system encoder. When the all-zeros code is present, adjust the pot for 0V at Pin 6 of the LF356.

A Monolithic Linear Drive Using LM12 Power Op Amp

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

Typical PWM Motor Drive Interfaces

Figure 16 shows an LM18298 dual full-bridge driver interfaced to the LM629 PWM outputs to provide a switch-mode power amplifier for driving small brush/commutator motors. *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 Gauss-ian-noise environments. If noise is added to the encoder inputs (either or both inputs) and is such that it is not sustained until the next encoder transition, the LM628 decoder logic will reject it. Noise that mimics quadrature counts or persists through encoder transitions must be eliminated by appropriate EMI design.

Simple digital "filtering" schemes merely reduce susceptibility to noise (there will always be noise pulses longer than the filter can eliminate). Further, any noise filtering scheme reduces decoder bandwidth. In the LM628 it was decided (since simple filtering does not eliminate the noise problem) to not include a noise filter in favor of offering maximum possible decoder bandwidth. Attempting to drive encoder signals too long a distance with simple TTL lines can also be a source of "noise" in the form of signal degradation (poor risetime and/or ringing). This can also cause a system to lose positional integrity. Probably the most effective countermeasure to noise induction can be had by using balanced-line drivers and receivers on the encoder inputs. *Figure 18* shows circuitry using the DS26LS31 and DS26LS32.

Typical Applications (Continued)

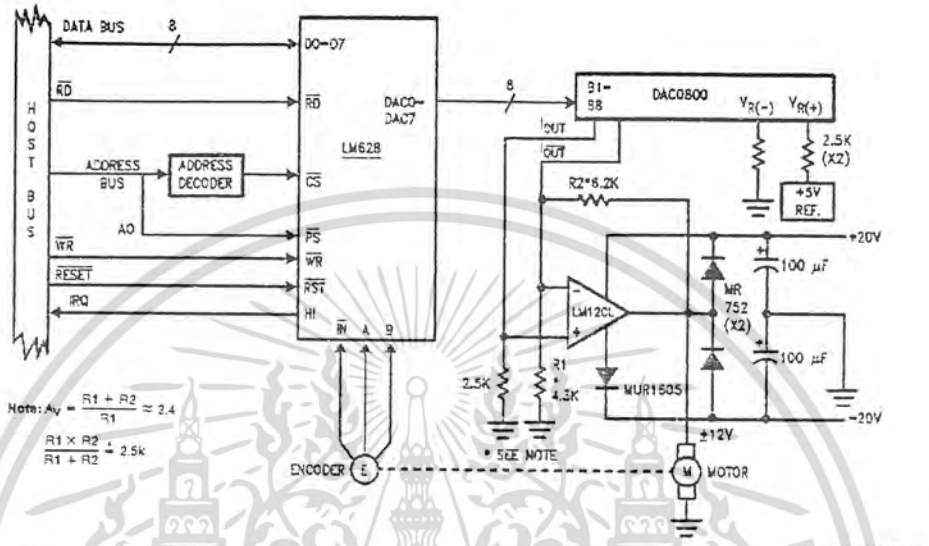


FIGURE 12. Host Interface and Minimum System Configuration

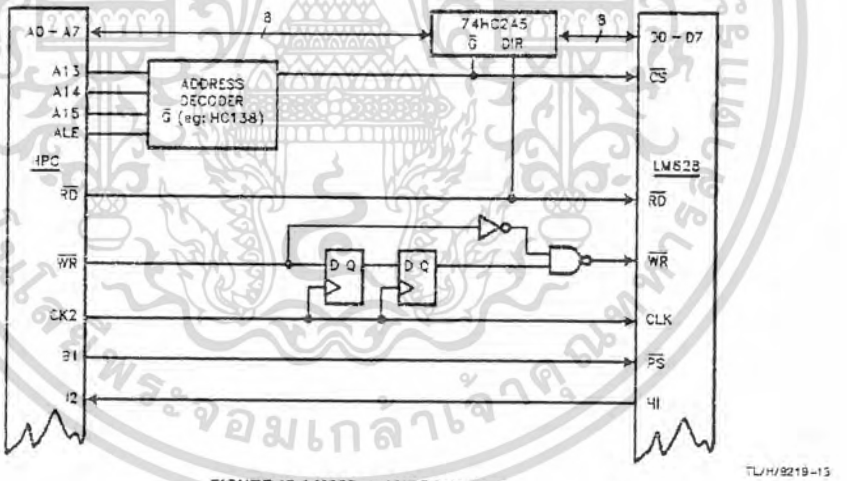
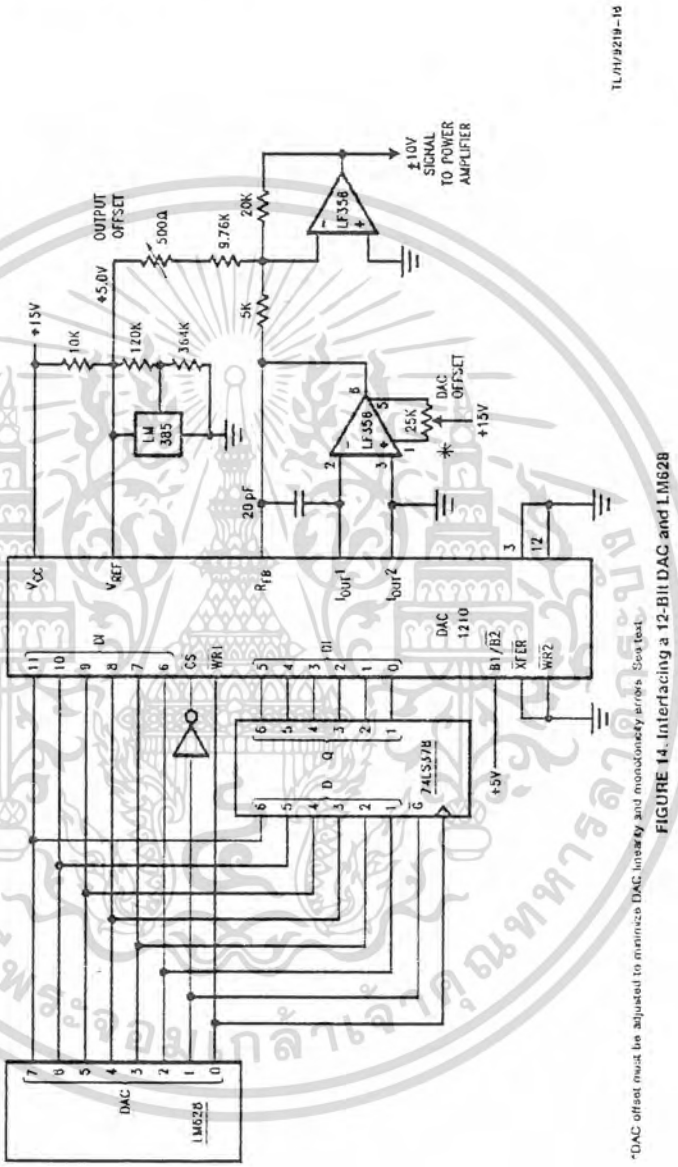


FIGURE 13. LM628 and HPC Interface

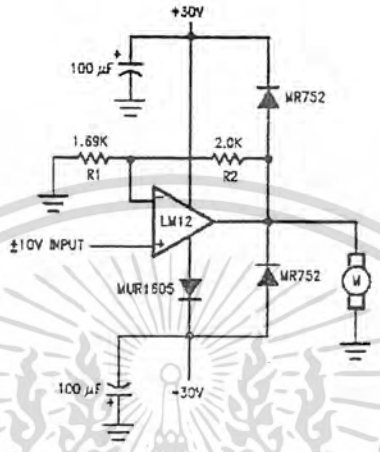
Typical Applications (Continued)



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

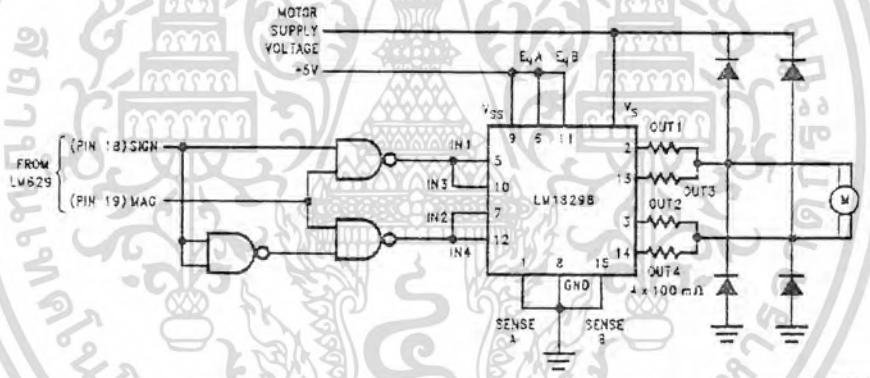
FIGURE 14. Interfacing a 12-Bit DAC and LM628

Typical Applications (Continued)



TL/H/9219–17

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



TL/H/9219–18

FIGURE 16. PWM Drive for Brush/Commutator Motors

Typical Applications (Continued)

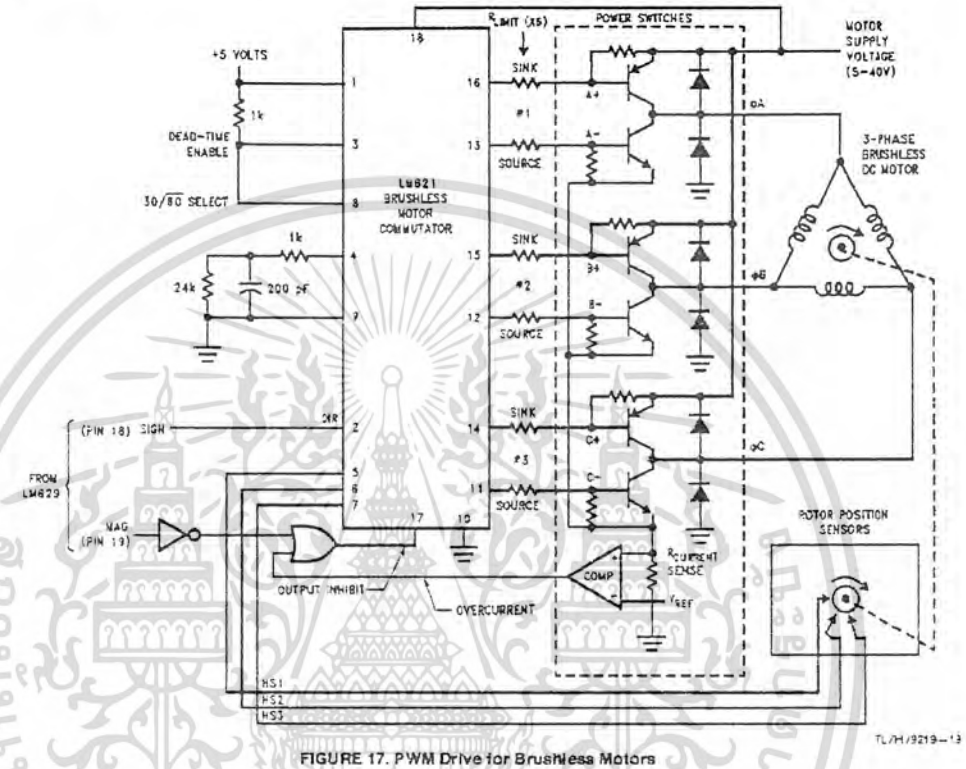


FIGURE 17. PWM Drive for Brushless Motors

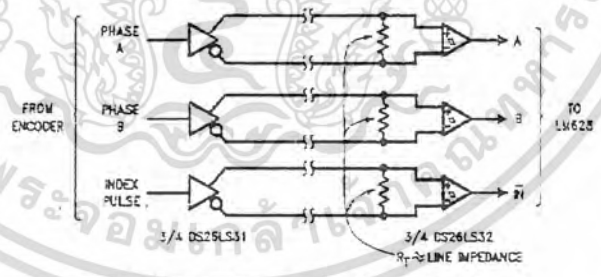
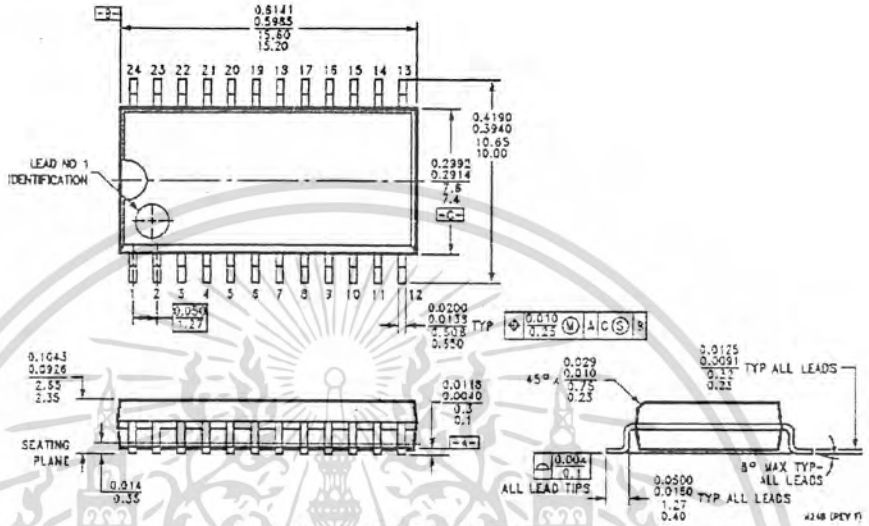


FIGURE 18. Typical Balanced-Line Encoder Input Circuit

Physical Dimensions inches (millimeters)



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

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.

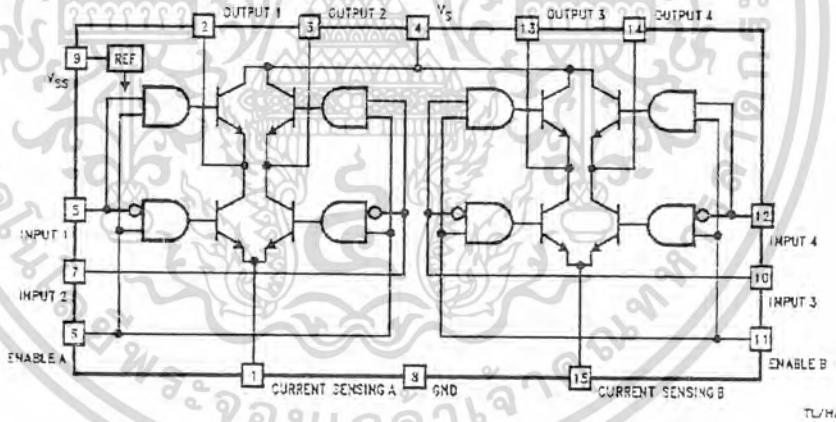
Features

- 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

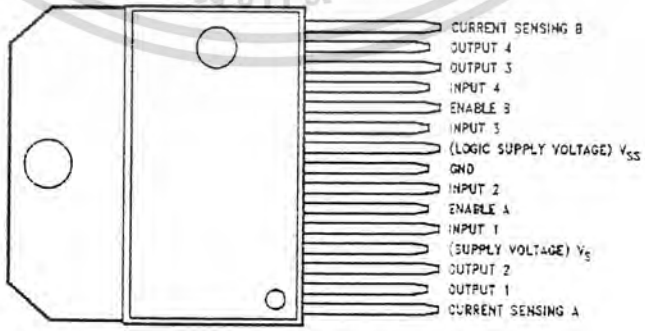
Applications

- DC and stepper motor drivers
- Relay and solenoid drivers

Block & Connection Diagrams



TL/H/9302–1



TL/H/9302–2

TO 220-15
Order Number LM18298T
NS Package Number TA15A

RRD-B30M115/Printed in U. S. A.

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	9	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	6	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)	$t < 50\ \mu s$		−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 $T_{J\,max}$, θ_{JC}, and T_C. The maximum allowable power dissipation at any temperature is $P_D = (T_{J\,max} - T_C)/\theta_{JC}$ or the number given in the Absolute Maximum Ratings, whichever is lower. The typical junction-to-case thermal resistance (θ_{JC}) of the LM18298 is 3°C/W.

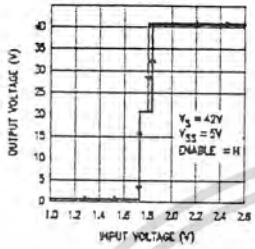
Note 3: Human body model, 100 pF discharged through a 1.5 kΩ resistor.

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

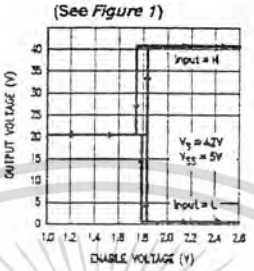
Note 5: Limits are guaranteed and 100% tested.

Typical Performance Characteristics

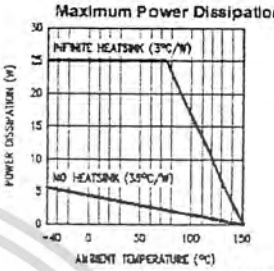
Output Voltage (See *Figure 1*)



Output Voltage (See *Figure 1*)

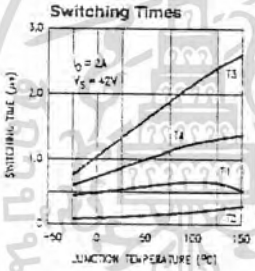


Maximum Power Dissipation

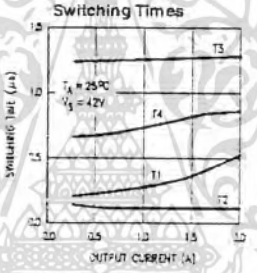


TL/H/9302–3

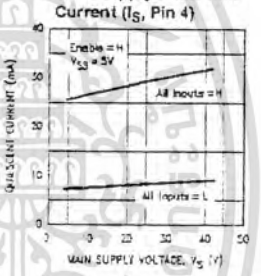
Source Current Switching Times



Source Current Switching Times

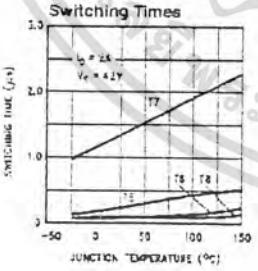


Main Supply Quiescent Current (I_S, Pin 4)

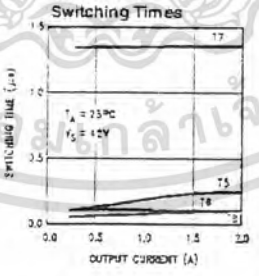


TL/H/9302–12

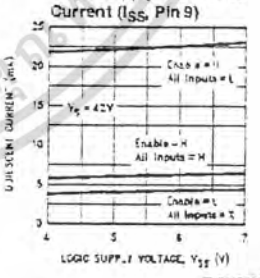
Sink Current Switching Times



Sink Current Switching Times

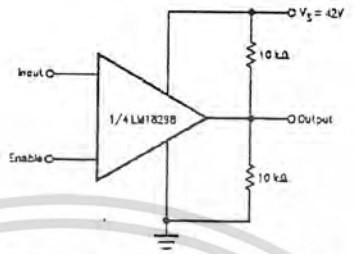


Logic Supply Quiescent Current (I_{SS}, Pin 9)



TL/H/9302–13

Test Circuits



TL/H/9302—4

FIGURE 1. Input/Enable Threshold Test Circuit

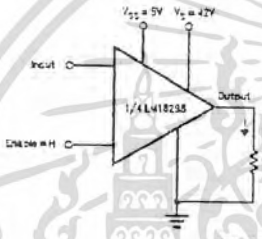


FIGURE 2(a). Source Current Switching Time Test Circuit

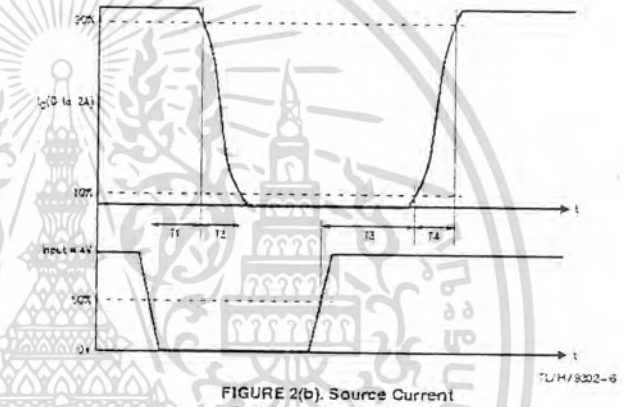


FIGURE 2(b). Source Current Switching Time Definitions

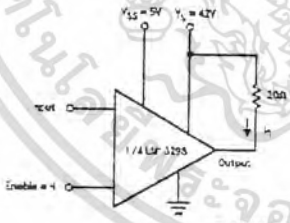


FIGURE 3(a). Sink Current Switching Time Test Circuit

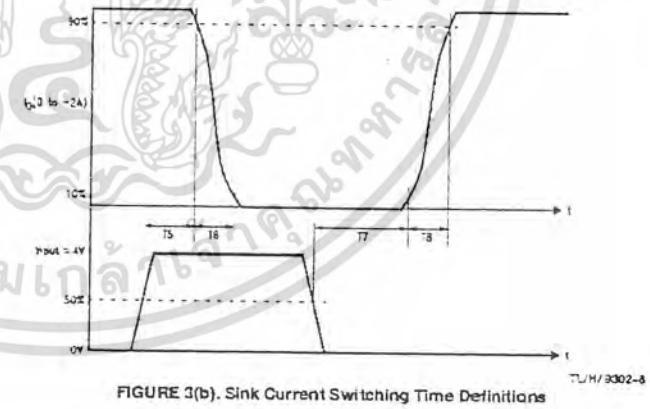
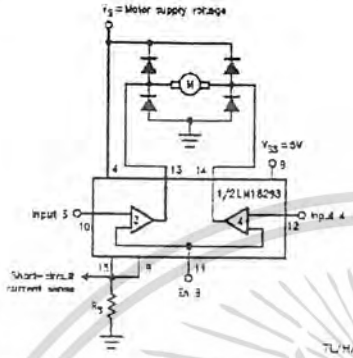


FIGURE 3(b). Sink Current Switching Time Definitions

Applications Information

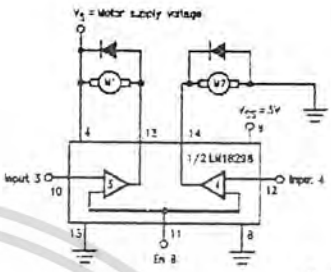


TL/H/9302–9

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

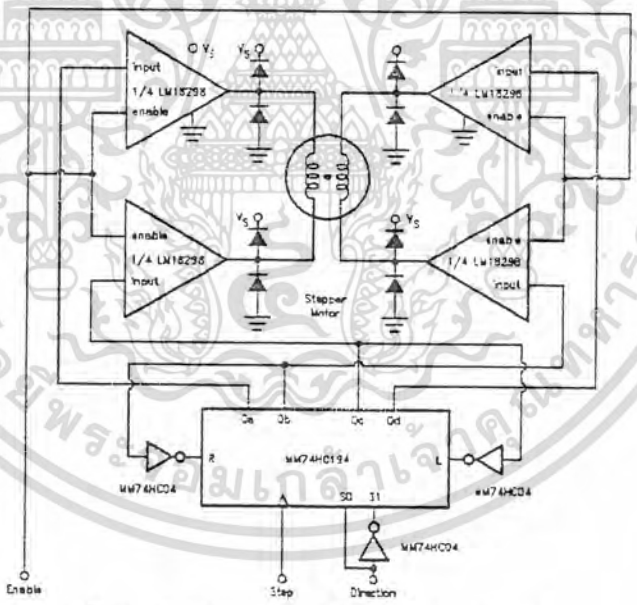


TL/H/9302–10

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)



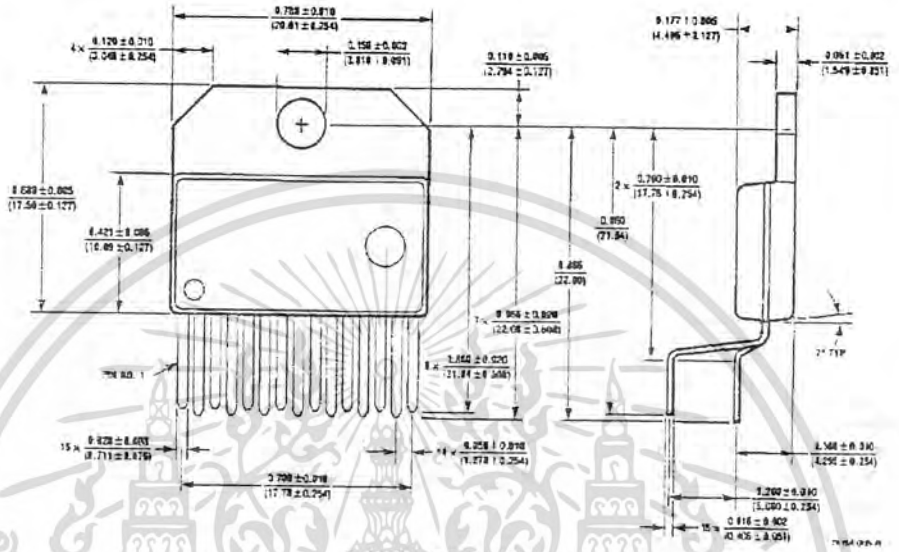
TL/H/9302–11

FIGURE 6. Two-Phase Bipolar Stepper Motor Control Circuit

CLAMP DIODES

When driving inductive loads, diodes are necessary to clamp spikes at the LM18298 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 ง

Source Code ของโปรแกรมที่ใช้ในการควบคุม

```
/* โปรแกรมทดสอบการควบคุมเมื่อใช้ตัวสังเกตสเตทที่ออกแบบจากคาล์มานฟิลเตอร์
   ชื่อไฟล์ "LQG.C"
        Begin        24 MARCH 1996
        Last Update  18 APRIL 1996
*/

/* Standard Header Files */
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <time.h>
#include <math.h>

/* Our Header Files */
#include <adc.h>
#include <lm629v.h>
#include <plotgp.h>

/* Define Constant Macro*/
#define PERIOD 0.1
define LOW1 2500                    //value to load to counter2 8253
#define LOW2 5000
#define PI  3.14159265359
#define X  16
#define Y  5
#define MAXDATA 600
#define cranefile "test.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;         //select PACER Mode to use as Sampling Time of Process
        int base=0x220;         //select address H220 to use as BASE Address
        int ch=12;              //select channel 12 to use
        counterRegister counter;
        float offset;
        float volt2angle=(30/3);        //value converse from voltage to angle in degree

        graphInfo info;
        axisScale scale;
        float maxDistance=1.0;  //Set Maximum Value In Y-Axis of Graph
        float samplingTime,maxTime=60.0;  //Set Maximum Value In X-Axis of Graph

        int k=0;
        long time=0;
        long startProcess=0,startLoop=0,end=0;

        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 difPos[MAXDATA],difDifPos[MAXDATA],difAngle[MAXDATA];
        float controlLaw[MAXDATA];

        feedBackGain Kf;
        float f1,f2,f3,f4,f5;
        float Ko[7][4],L[7][4];
        float G[7][7],H[7][3];
        float errorPos[MAXDATA],errorAngle[MAXDATA];

        float estimAngle[MAXDATA],estimPos[MAXDATA],estimDifAngle[MAXDATA];
        float estimDifPos[MAXDATA],estimDifDifPos[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=1.0;                    //Set Value of POSITION REFERENCE
        float angleRef=0.0;                  //Set Value of ANGLE REFERENCE

/* Start Program */
        clrscr();

/* Set 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;              /* Voltage Reading from ADC On PCL-714 Card */
            realAngle[k]=0;         /* Real Angle Value Converse from Voltage */
            realPos[k]=0;           /* Real Position Value Reading from LM629 */
            realDifAngle[k]=0;      /* Value Compute On Physics Relation */
            realDifPos[k]=0;        /* Value Compute On Physics Relation */
            realDifDifPos[k]=0;     /* Value Compute On Physics Relation */
            angle[k]=0;             /* Value After Measurement Update */
            difAngle[k]=0;          /* Value After Measurement Update */
            pos[k]=0;               /* Value After Measurement Update */
            difPos[k]=0;            /* Value After Measurement Update */
            difDifPos[k]=0;         /* Value After Measurement Update */
            estimAngle[k]=0;        /* Value Before Measurement Update */
            estimPos[k]=0;          /* Value Before Measurement Update */
            estimDifAngle[k]=0;     /* Value Before Measurement Update */
            estimDifPos[k]=0;       /* Value Before Measurement Update */
            estimDifDifPos[k]=0;    /* Value Before Measurement Update */
            controlLaw[k]=0;        /* Control Law */
            errorPos[k]=0;
            errorAngle[k]=0;
            ePos[k]=0;              /* realPos-estimatePos */
            eAngle[k]=0;            /* realAngle-estimateAngle */
            filAngle[k]=0;      /* Angle Value That has Filtered from Band Pass Filter */
        }
```

```
/* Initial FeedBack Gain */
/* Calculate from Q=diag([100 1 1000 1 1]) ,R=diag([100 100])  (see Example in chaper 5) */
        Kf.angle=-0.3272;
        Kf.difAngle=-0.1656;
        Kf.pos=1.9871;
        Kf.difPos=0.8480;
        Kf.difDifPos=0.1549;

/* State Matrix (G) */
        /* Sampling Period = 0.1 sec */
        G[1][1]=0.9393; G[1][2]=0.0980; G[1][3]=0.0000; G[1][4]=0.0025; G[1][5]=-0.0050;
        G[2][1]=-1.2013;G[2][2]=0.9393; G[2][3]=0.0000; G[2][4]=0.0721; G[2][5]=-0.0894;
        G[3][1]=0.0000; G[3][2]=0.0000; G[3][3]=1.0000; G[3][4]=0.0980; G[3][5]=0.0041;
        G[4][1]=0.0000; G[4][2]=0.0000; G[4][3]=0.0000; G[4][4]=0.9417; G[4][5]=0.0733;
        G[5][1]=0.0000; G[5][2]=0.0000; G[5][3]=0.0000; G[5][4]=-1.0469;G[5][5]=0.4985;

/* Input Matrix (H) */
        /* Sampling Period = 0.1 sec */
        H[1][1]=-0.0025;
        H[2][1]=-0.0721;
        H[3][1]=0.0020;
        H[4][1]=0.0583;
        H[5][1]=1.0469;

/* Kalman Filter Gain (Observer Gain.Ko) */
/* Calculate from Qf11=diag([100 100 100 100 100]), Rf11=diag([0.01 0.01]) ,Ts=0.1 */
        Ko[1][1]=0.9911; Ko[1][2]=0.0001;
        Ko[2][1]=0.7705; Ko[2][2]=0.1268;
        Ko[3][1]=0.0001; Ko[3][2]=0.9905;
        Ko[4][1]=0.0821; Ko[4][2]=0.2517;
        Ko[5][1]=-0.3222;Ko[5][2]=-0.4849;
```

```
/* Band-Pass Filter for filter Noise from Angle */
        a1[1]=1.0000; a1[2]=0.0309; a1[3]=-0.9691;
        b1[1]=0.9845; b1[2]=0.0000; b1[3]=-0.9845;

        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
        printf("\n\nPress any key to start test.");
        getch( );

/* Initial LM629 */
        INIT_LM629( );                            //initial lm629
        WRITE_FILTER( );
        ACCELERATION_OUT(8);                      //initial acceleration
/* Initial Graphic */
        scale = INITIAL_GRAPH(info);              //open graphic mode and initial to plot graph
/* Initial 8253 Counter */
        SET_COUNTER(base,counter);                //load sampling time and start counting

        startProcess=clock();                     //read start Process clock
        startLoop=startProcess;                   //read start Loop clock
/* Start Loop Control */
        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 sampling 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;  //from voltage to "radian"

/* Filter Angle before Compute */
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

/* State Estimator Calculate DifAngle DifPos DifDifPos
  From Physics Relation for LQR */
if (k==1)
{
        realDifPos[k]=0;
```

```
                realDifDifPos[k]=0;
                realDifAngle[k]=0;
        }
        else
        {
                realDifPos[k]=(realPos[k]-realPos[k-1])/info.sampling;
                realDifDifPos[k]=(realDifPos[k]-realDifPos[k-1])/info.sampling;
                realDifAngle[k]=(realAngle[k]-realAngle[k-1])/info.sampling;
        }
/*      if (k==2)
        {
                realDifDifPos[k]=0;
        }
        else
        {
                realDifDifPos[k]=(realDifPos[k]-realDifPos[k-1])/info.sampling;
        }
*/
        /* State Estimator Design on Kalman Filer For LQG */
        /* Time Update */
        angle[k]=(G[1][1]*angle[k-1])+(G[1][2]*difAngle[k-1])+(G[1][3]*pos[k-1])+(G
[1][4]*difPos[k-1])+(G[1][5]*difDifPos[k-1])+(H[1][1]*controlLaw[k-1]);
        difAngle[k]=(G[2][1]*angle[k-1])+(G[2][2]*difAngle[k-1])+(G[2][3]*pos[k-1])
+(G[2][4]*difPos[k-1])+(G[2][5]*difDifPos[k-1])+(H[2][1]*controlLaw[k-1]);
        pos[k]=(G[3][1]*angle[k-1])+(G[3][2]*difAngle[k-1])+(G[3][3]*pos[k-1])+ (G
[3][4]*difPos[k-1])+(G[3][5]*difDifPos[k-1])+(H[3][1]*controlLaw[k-1]);
        difPos[k]=(G[4][1]*angle[k-1])+(G[4][2]*difAngle[k-1])+(G[4][3]*pos[k-1])+
(G[4][4]*difPos[k-1])+(G[4][5]*difDifPos[k-1])+(H[4][1]*controlLaw[k-1]);
        difDifPos[k]=(G[5][1]*angle[k-1])+(G[5][2]*difAngle[k-1])+(G[5][3]*
pos[k-1])+(G[5][4]*difPos[k-1])+(G[5][5]*difDifPos[k-1])+(H[5][1]*controlLaw[k-1]);
```

```
        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);
/* Write Data to File */
        datafile=fopen(cranefile,"wt");            //open file to write data
        gotoxy(X-6,Y+2);
        printf("Wait for writing data to file.");
        for (k=0;k<MAXDATA;k++)
        {
        /* Write data to file */
        fprintf(datafile,"%2.8f %2.8f %2.8f %2.8f %2.8f %2.8f %2.8f %2.8f %2.8f %2.8f
%2.8f %2.8f %2.8f %2.8f %2.8f %2.8f %2.8f \n",controlLaw[k],filAngle[k],realAngle[k],
realDifAngle[k],realPos[k],realDifPos[k],realDifDifPos[k],angle[k],difAngle[k],pos[k],difPos[k],
difDifPos[k],estimAngle[k],estimDifAngle[k],estimPos[k],estimDifPos[k],estimDifDifPos[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( );

}       //End Of Program
```

```
/* โปรแกรมทดสอบการควบคุมเมื่อใช้ตัวสังเกตสเตทที่ใช้ความสัมพันธ์ทางฟิสิกส์
   ชื่อไฟล์ "LQR.C"
        Begin        24 MARCH 1996
        Last Update  18 APRIL 1996
*/

/* Standard Header Files */
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <time.h>
#include <math.h>

/* Our Header Files */
#include <adc.h>
#include <lm629v.h>
#include <plotgp.h>

/* Define Constant Macro*/
#define PERIOD 0.1
define LOW1 2500                    //value to load to counter2 8253
#define LOW2 5000
#define PI  3.14159265359
#define X  16
#define Y  5
#define MAXDATA 600
#define cranefile "test.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;         //select PACER Mode to use as Sampling Time of Process
        int base=0x220;         //select address H220 to use as BASE Address
        int ch=12;              //select channel 12 to use
        counterRegister counter;
        float offset;
        float volt2angle=(30/3);        //value converse from voltage to angle in degree

        graphInfo info;
        axisScale scale;
        float maxDistance=1.0;          //Set Maximum Value In Y-Axis of Graph
        float samplingTime,maxTime=60.0;  //Set Maximum Value In X-Axis of Graph

        int k=0;
        long time=0;
        long startProcess=0,startLoop=0,end=0;

        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 difPos[MAXDATA],difDifPos[MAXDATA],difAngle[MAXDATA];
        float controlLaw[MAXDATA];

        feedBackGain Kf;
        float f1,f2,f3,f4,f5;
        float Ko[7][4],L[7][4];
        float G[7][7],H[7][3];
        float errorPos[MAXDATA],errorAngle[MAXDATA];

        float estimAngle[MAXDATA],estimPos[MAXDATA],estimDifAngle[MAXDATA];
        float estimDifPos[MAXDATA],estimDifDifPos[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=1.0;                    //Set Value of POSITION REFERENCE
        float angleRef=0.0;                  //Set Value of ANGLE REFERENCE

/* Start Program */
        clrscr();

/* Set 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;              /* Voltage Reading from ADC On PCL-714 Card */
            realAngle[k]=0;         /* Real Angle Value Converse from Voltage */
            realPos[k]=0;           /* Real Position Value Reading from LM629 */
            realDifAngle[k]=0;      /* Value Compute On Physics Relation */
            realDifPos[k]=0;        /* Value Compute On Physics Relation */
            realDifDifPos[k]=0;     /* Value Compute On Physics Relation */
            angle[k]=0;             /* Value After Measurement Update */
            difAngle[k]=0;          /* Value After Measurement Update */
            pos[k]=0;               /* Value After Measurement Update */
            difPos[k]=0;            /* Value After Measurement Update */
            difDifPos[k]=0;         /* Value After Measurement Update */
            estimAngle[k]=0;        /* Value Before Measurement Update */
            estimPos[k]=0;          /* Value Before Measurement Update */
            estimDifAngle[k]=0;     /* Value Before Measurement Update */
            estimDifPos[k]=0;       /* Value Before Measurement Update */
            estimDifDifPos[k]=0;    /* Value Before Measurement Update */
            controlLaw[k]=0;        /* Control Law */
            errorPos[k]=0;
            errorAngle[k]=0;
            ePos[k]=0;              /* realPos-estimatePos */
            eAngle[k]=0;            /* realAngle-estimateAngle */
            filAngle[k]=0;      /* Angle Value That has Filtered from Band Pass Filter */
        }
```

```
/* Initial FeedBack Gain */
/* Calculate from Q=diag([100 1 1000 1 1]) ,R=diag([100 100])  (see Example in chaper 5) */
        Kf.angle=-0.3272;
        Kf.difAngle=-0.1656;
        Kf.pos=1.9871;
        Kf.difPos=0.8480;
        Kf.difDifPos=0.1549;

/* State Matrix (G) */
        /* Sampling Period = 0.1 sec */
        G[1][1]=0.9393; G[1][2]=0.0980; G[1][3]=0.0000; G[1][4]=0.0025; G[1][5]=-0.0050;
        G[2][1]=-1.2013;G[2][2]=0.9393; G[2][3]=0.0000; G[2][4]=0.0721; G[2][5]=-0.0894;
        G[3][1]=0.0000; G[3][2]=0.0000; G[3][3]=1.0000; G[3][4]=0.0980; G[3][5]=0.0041;
        G[4][1]=0.0000; G[4][2]=0.0000; G[4][3]=0.0000; G[4][4]=0.9417; G[4][5]=0.0733;
        G[5][1]=0.0000; G[5][2]=0.0000; G[5][3]=0.0000; G[5][4]=-1.0469;G[5][5]=0.4985;

/* Input Matrix (H) */
        /* Sampling Period = 0.1 sec */
        H[1][1]=-0.0025;
        H[2][1]=-0.0721;
        H[3][1]=0.0020;
        H[4][1]=0.0583;
        H[5][1]=1.0469;

/* Kalman Filter Gain (Observer Gain,Ko) */
/* Calculate from Qf11=diag([100 100 100 100 100]), Rf11=diag([0.01 0.01]) ,Ts=0.1 */
        Ko[1][1]=0.9911; Ko[1][2]=0.0001;
        Ko[2][1]=0.7705; Ko[2][2]=0.1268;
        Ko[3][1]=0.0001; Ko[3][2]=0.9905;
        Ko[4][1]=0.0821; Ko[4][2]=0.2517;
        Ko[5][1]=-0.3222;Ko[5][2]=-0.4849;
```

```
/* Band-Pass Filter for filter Noise from Angle */
        a1[1]=1.0000; a1[2]=0.0309; a1[3]=-0.9691;
        b1[1]=0.9845; b1[2]=0.0000; b1[3]=-0.9845;

        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
        printf("\n\nPress any key to start test.");
        getch( );

/* Initial LM629 */
        INIT_LM629( );                            //initial lm629
        WRITE_FILTER( );
        ACCELERATION_OUT(8);                      //initial acceleration
/* Initial Graphic */
        scale = INITIAL_GRAPH(info);              //open graphic mode and initial to plot graph
/* Initial 8253 Counter */
        SET_COUNTER(base.counter);                //load sampling time and start counting

        startProcess=clock();                     //read start Process clock
        startLoop=startProcess;                   //read start Loop clock
/* Start Loop Control */
        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 sampling 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;  //from voltage to "radian"

/* Filter Angle before Compute */
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

/* State Estimator Calculate DifAngle DifPos DifDifPos
   From Physics Relation for LQR */
if (k==1)
{
        realDifPos[k]=0;
```

```
            realDifDifPos[k]=0;
            realDifAngle[k]=0;
        }
        else
        {
            realDifPos[k]=(realPos[k]-realPos[k-1])/info.sampling;
            realDifDifPos[k]=(realDifPos[k]-realDifPos[k-1])/info.sampling;
            realDifAngle[k]=(realAngle[k]-realAngle[k-1])/info.sampling;
        }
/*      if (k==2)
        {
            realDifDifPos[k]=0;
        }
        else
        {
            realDifDifPos[k]=(realDifPos[k]-realDifPos[k-1])/info.sampling;
        }
*/
        errorAngle[k]=angleRef - filAngle[k];
        errorPos[k]=posRef - realPos[k];
        /* Compute Control Law (Velocity Command) */
        /* Estimate State from Physic Relation */
        f2=realDifAngle[k]*Kf.difAngle;
        f3=realDifPos[k]*Kf.difPos;
        f4=realDifDifPos[k]*Kf.difDifPos;
        controlLaw[k]=(Kf.angle*errorAngle[k])+(Kf.pos*errorPos[k])-(f2+f3+f4);

        /* Send Velocity Command to Speed Control */
        VELOCITY_OUT(controlLaw[k]);        //send command to Speed Control

        /* Display Position and Angle on Graph */
        UPDATE_GRAPH(scale,realPos[k],realAngle[k],samplingTime);
```

```
            samplingTime=samplingTime+info.sampling;      //update new time
        }while(samplingTime<maxTime);                     //End Loop Control
        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);
/* Write Data to File */
        datafile=fopen(cranefile,"wt+");                  //open file to write data
        gotoxy(X-6,Y+2);
        printf("Wait for writing data to file.");
        for (k=0;k<MAXDATA;k++)
        {
        /* Write data to file */
        fprintf(datafile,"%2.8f %2.8f %2.8f %2.8f %2.8f %2.8f %2.8f \n",controlLaw[k],
filAngle[k],realAngle[k],realDifAngle[k],realPos[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( );

}       //End Of Program
```

```
/* Source Code Of Header File ,Name "ADC.H" */
/* 14 bit Analog to Digital Conversion (in PCL-714 Broad)
   Creat BY  Mr.SUNIBOON TUNGWINYOO
   Begin          5 FEB 1996
   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                         //defult 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
```

```
/* Prototype of Function */

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;

        low=(int)sqrt(period);                  //define low word
        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);
}

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;

        /* 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
        */

        do
        {
            /* wait for complete conversion */
                data.high=inp(base-5);          //read high_byte data
        }while(data.high>=64);                  //if hi >=64 mean conversion is not complete
        data.low=inp(base-4);                   //read low_byte data

        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
        outp(base+11,mode);                     //select mode
}
```

```
/* Source Code of Header File of Graph Ploting
   File Name = " PLOTGP.H" */
/* Plot Graph In Graphic Mode
   Creat BY  Mr.SUNIBOON TUNGWINYOO
   Begin          5 FEB 1996
   Last Update    30 MAR 1996
*/

#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)
```

```
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 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 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 */

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));
                exit(1);
        }
        /* Initialize graphics */
        initgraph(&gdrive,&gmode,"");
        /* Check error */
        errorcode = graphresult();
        if(errorcode != grOk) /* An error occurred */
        {
                printf("Error while opening \n");
                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 50 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
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
        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 cm",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 angle,float time)
{
        int line=10;                                    //line of text (text heigth)
        char buffer[20];                                //buffer of data
        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);
        bar(LEFT_DESCRIP+150,TOP_DESCRIP+(1.5*line),LEFT_DESCRIP+250,TOP_DE
SCRIP+(2.5*line));                                      //clear area to write new data
```

```
//*************************************************************************
//*************************************************************************
//********                    LM629V.H                           ********
//********                LM629 HEADER FILE                      ********
//********                                                       ********
//*************************************************************************
//*************************************************************************

#define ADDR_COM          0X300       //write command and read
                                      //status flag address
#define ADDR_DATA         0X301       //read and write data address
#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*341.3E-6*65536
#define veloConvertM2C    744617.585  //multiplier constant for convert
                                      //velocity m/s to count per sec
                                      //33287*341.3E-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*(341.3E-6)^2*65536
#define accConvertC2m     3.934498E-3 //multiplier constant for convert
                                      //acceleration count/sec^2 to m/s^2
                                      //1/33287*(341.3E-6)^2*65536
```



```
#define kpConvert         256         //kp=256*Kp
```

```
#define kiConvert              2.8          //ki=65536*Ki*T
#define kdConvert              750000       //kd=256*Kd/Ts
#define kp                     3.48         //propertion gain
#define ki                     4.43         //integration gain
#define kd                     0.68         //derivative gain
#define il                     50000        //integration limit
#define filterCommand          0x000F       //load filter parametor command
#define InitMaxAccCommand      0x1820       //trajectory control word a 18 hex in
                                            //High Byte mean forward direction
                                            //velocity mode and acceleration
                                            //will be loaded
#define maxVelocity            1.2          //maximum velocity in m/s
#define maxAcceleration        4            //maximum acceleration in m/s^2
#define posCommand             0x002A       //position mode ACC VELO position will
                                            //be loaded and it 's absolute
#define forwVeloCommand        0x1808       //it's mean forward direction
                                            //velocity will be loaded
#define rewVeloCommand         0x0808       //it's mean reward direction
                                            //velocity will be load loaded
#define stopMotorCommand       0x0200       //it's stop motor abruptly

#define resetBreakCommand      0x0000       //it's mean 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
#define MSKI                   0x1C         //mask interrupts
```

```
#define RSTI        0x1D    //reset interrupts
#define LFIL        0x1E    //load filter parameter
#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 busy
void WRITE_COM(unsigned char);          //write command
void WRITE_DATA_BYTE(unsigned char);    //write data 1 byte
void WRITE_DATA_2BYTES(int);            //write data 2 bytes
void WRITE_DATA_4BYTES(long);           //write data 4 bytes
void WRITE_DATA_WORD(int);      //write data word ONLY FILTER DATA
void WRITE_DATA_2WORDS(long);   //write 2 data words ONLY TRAJECTORY DATA
unsigned char READ_DATA_BYTE(void);     //read 1 data byte
int  READ_DATA_2BYTES(void);            //read 2 data bytes
long READ_DATA_4BYTES(void);            //read 4 data bytes
void SET_BREAK_POINT(float);            //set break point
void INIT_LM629(void);                  //initial lm629
void WRITE_FILTER(void);                //write PID filter parameter
void ACCELERATION_OUT(float);           //write maximum acceleration
void POSITION_OUT(float);               //write desired position command
void VELOCITY_OUT(float);               //write desired velocity command
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 until busy bit 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 command
        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);                   //software reset
        delay(500);

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

```
void WRITE_FILTER(void)
{

        unsigned int kpPulse,kiPulse,kdPulse,ilPulse;
        WRITE_COM(LFIL);                        //load filter command
        WRITE_DATA_WORD(filterCommand);         //control word and derivative time

        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
}

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

        WRITE_DATA_WORD(InitMaxAccCommand); //write control word
                                                //velocity mode acceleration
                                                //will be loaded
        WRITE_DATA_2WORDS((long)(acc*accConvertM2C));
        WRITE_COM(STT);                         //start trajectory
```

```
}

void POSITION_OUT(float pos)                    //write desired position
{
        WRITE_COM(LTRJ);
        WRITE_DATA_WORD(posCommand);
        WRITE_DATA_2WORDS((long)(maxAcceleration*accConvertM2C));
        WRITE_DATA_2WORDS((long)(maxVelocity*veloConvertM2C));
        WRITE_DATA_2WORDS((long)(pos*posConvertM2C));
        WRITE_COM(STT);                         //start trajectory
}

void VELOCITY_OUT(float velocity)               //write desired 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)              //stop motor if reach break point
{
    if(0x40 & inp(ADDR_COM))                //stop motor abruptly
        {
        WRITE_COM(LTRJ);
        WRITE_DATA_WORD(stopMotorCommand);
        WRITE_COM(STT);
        WRITE_COM(RSTI);                    //reset interrupt
        WRITE_DATA_2BYTES(resetBreakCommand);   //reset all interrupt
        }
}

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
}


//*****************************************************************************
//                          END PROGRAM
//*****************************************************************************
```

หนังสืออ้างอิง

1. รศ. วิพันธ์ ปรีชาพานิช , การวิเคราะห์ระบบควบคุมเวลาดีสครีต , คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง , 2536
2. รศ. วิพันธ์ ปรีชาพานิช , ระบบควบคุมออพติมัล , คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง , 2532
3. ผศ.ดร. โยธิน เปรมปรีชาพานิช,วิเคราะห์และออกแบบ ระบบควบคุมมอเตอร์ ,คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง , 2526
4. ธันวา ศรีประโมง , การเขียนโปรแกรมภาษาซีสำหรับวิศวกรรม , มหาวิทยาลัยเทคโนโลยีมหานคร , 2537
5. ธานินทร์ ถาวรศาสนวงศ์ , ทินกร ดุ๊ก , การอินเตอร์เฟส IBM PC , ฟิสิกส์เซนเตอร์ , 2536
6. Frank L. Lewis , *Applied Optimal Control & Estimation* , Prentic-Hall International ,Inc. ,1992.
7. Katsuhiko Okata , *Model Control Engineering* , Prentic-Hall International ,Inc. , 1990.
8. Katsuhiko Okata , *Design Linear Control Systems with MATLAB* , Prentic-Hall International ,Inc. , 1994.
9. Hans Butler , *Model Referance Adaptive Control* , Prentic-Hall Internation ,Inc. , 1992.
10. Bahram Shahian , Michael Hassul , *Control System Design Using MATLAB* , Prentic-Hall International ,Inc. ,1993.
11. The Math Works ,Inc. , *PC-MATLAB* ,The Math Works ,Inc. ,October 15,1990.
12. The Math Works ,Inc. , *Control System Toolbox* ,The Math Works ,Inc.,October 30,1990.
13. The Math Works ,Inc. , *SIMULINK* , The Math Works ,Inc. , April 1993.
14. The Math Works ,Inc. ,*Signal Processing Toolbox*. John Little and Loren Shure , The Math Works ,Inc. ,August 28,1988.
15. Noriyuki Komine , *Optimal Control of Anti-Sway for Overhead Traveling Crane* , Tokai University ,1995