ระบบกล้องวีดีโอวงจรปิดติดตามวัตถุอัตโนมัติ

AUTOMATIC TRACKING SECURITY CAMERA SYSTEM

โดย

นายภูริธัช แท่นมณี

นายภูษิต มุ่งมานะกิจ

นายมนตรี จูสี่พร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ระบบกล้องวีดีโอวงจรปิดติดตามวัตถุอัตโนมัติ

AUTOMATIC TRACKING SECURITY CAMERA SYSTEM

โดย

นายภูริธัช แท่นมณี 37014325

นายภูษิต มุ่งมานะกิจ 37014328

นายมนตรี จูสี่พร 37014332

อาจารย์ที่ปรึกษา

ผศ.ดร.ไกรสิน ส่งวัฒนา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์ปีการศึกษา 2540

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบกล้องวีดีโอวงจรปิดติดตามวัตถุอัตโนมัติ

Automatic tracking security camera system

ผู้จัดทำ 1. นาย ภูริธัช แท่นมณี 37014325

2. นาย ภูษิต มุ่งมานะกิจ 37014328

3. นาย มนตรี ภูสี่พร 37014332

ไกรสิน ส่งวัฒนา

..... อาจารย์ที่ปรึกษา

(ผศ. ดร. ไกรสิน ส่งวัฒนา)

ระบบกล้องวีดีโอวงจรปิดติดตามวัตถุอัตโนมัติ

Automatic tracking security camera system

โดย นายภูริธัช แท่นมณี 37014325
นายภูษิต มุ่งมานะกิจ 37014328
นายมนตรี จูสี่พร 37014332

อาจารย์ที่ปรึกษา ผศ. ดร. ไกรสิน ส่งวัฒนา

บทคัดย่อ

ระบบกล้องวีดีโอวงจรปิดเป็นระบบที่ใช้งานใน ด้านการรักษาความปลอดภัยอย่างกว้างขวาง โดยในโครงงานนี้จะทำการพัฒนาขีดความสามารถของระบบกล้องวีดีโอแบบเดิมโดยอาศัยเครื่องคอมพิวเตอร์และการเขียนโปรแกรมเพื่อควบคุมระบบกล้องวีดีโอ ทำให้สามารถทำการติดตามการเคลื่อนไหวของวัตถุ รวมถึงการจับภาพและการกำหนดอัตราการบันทึกภาพที่ได้จากกล้องวีดีโอ

ในส่วนของการเขียนโปรแกรมควบคุมระบบกล้องวีดีโอวงจรปิดจะใช้ภาษา C++ สำหรับส่วนที่ทำหน้าที่ในการควบคุมการหมุนของกล้องเพื่อติดตามวัตถุที่เคลื่อนที่จะอาศัย stepping motor โดยการควบคุมผ่านทางไมโครโปรเซสเซอร์ตระกูล MCS-51

Abstract

The security camera system is widely used in security system. This project is developed from the formal system with uses computer and programming to control the system that can track the moving object, including capture image and set saving rate.

In the part of programming for control task we use C++ language. For the task about control the rotate camera we use MCS-51 to control stepping motor.

สารบัญ

บทที่ 1
บทนำ

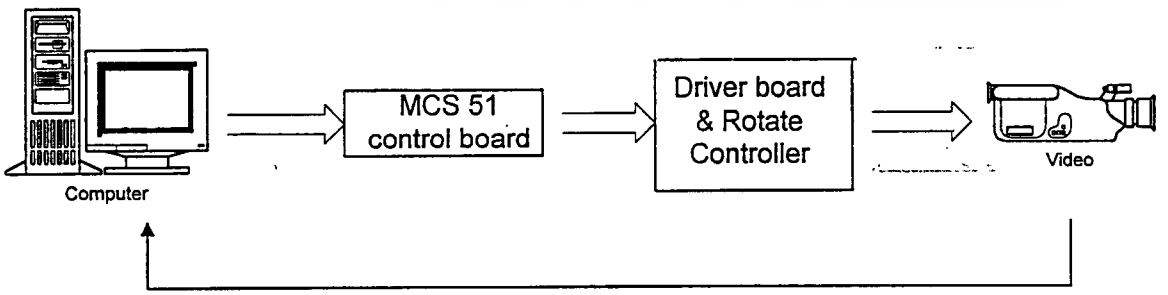
ระบบกล้องวีดีโอวงจรปิดติดตามวัตถุอัตโนมัติ เป็นการประยุกต์มาจากระบบกล้องวีดีโอวงจรปิดที่ใช้ในระบบรักษาความปลอดภัยทั่วๆ ไป ซึ่งในระบบเก่านั้นถ้าหากเราใช้กล้องวีดีโอเพียงตัวเดียวเราจะไม่สามารถทำการตรวจจับพื้นที่ได้กว้างพอหรือไม่ก็ต้องใช้จำนวนกล้องหลายตัวเพื่อให้ครอบคลุมบริเวณดังกล่าวได้ ซึ่งเป็นการสูญเสียเงินจำนวนมาก แต่ในปริญญานิพนธ์นี้ได้ทำการพัฒนาขึ้นโดยอาศัยความสามารถของเครื่องคอมพิวเตอร์ส่วนบุคคลที่ปัจจุบันมีความสามารถสูงมาประยุกต์ใช้งานกับกล้องวีดีโอขนาดเล็กราคาถูกซึ่งอาศัยการควบคุมจากโปรแกรมที่เขียนขึ้นโดยภาษา C++ และบอร์ดควบคุมขนาดเล็กที่ใช้ IC MCS-51 สำหรับการควบคุมทิศทางการจับภาพของกล้อง ทำให้เราสามารถใช้กล้องวีดีโอเพียงตัวเดียวทำการตรวจจับบริเวณได้มากขึ้นอีกทั้งจากระบบปฏิบัติการ windows 95 ที่รองรับการทำงานแบบ multi tastking ทำให้เราสามารถใช้คอมพิวเตอร์เพียงเครื่องเดียวแต่สามารถตรวจจับบริเวณต่างๆ ได้หลายที่ (หมายถึงกรณีที่มีการใช้กล้องหลายตัวเพื่อทำการตรวจจับในบริเวณต่างๆ) รวมถึงการใช้ hard disk ที่มีความคงทนในการเก็บบันทึกภาพวีดีโอ(เราสามารถควบคุมอัตราการบันทึกเพื่อเป็นการประหยัดเนื้อที่ในการเก็บข้อมูล) แทนการใช้สื่อการบันทึกที่เป็นเทปแบบเก่าได้อีกด้วย

1.1 ขอบเขตของปริญญานิพนธ์

1. การแสดงผลการจับภาพเป็นแบบ Real time mode ซึ่งสามารถควบคุมอัตราเร็วในการแสดงผลได้ (frames per second)
2. กล้องสามารถหมุนตามการเคลื่อนที่วัตถุที่มีรูปร่างใดๆ ได้ ในบริเวณที่มีฉากซึ่งไม่อยู่ในระยะอนันต์
3. ทำการบันทึกสัญญาณวีดีโอพร้อมทั้งทำการติดตามการเคลื่อนที่ของวัตถุได้

1.2 แนวความคิดเกี่ยวกับการทำงานของระบบกล้องวีดีโอวงจรปิดติดตามวัตถุอัตโนมัติ

ในหัวข้อนี้จะขอกล่าวถึงขั้นตอนการทำงานของปริญญานิพนธ์นี้คร่าวๆ โดยอาศัยการอธิบายตามการแสดงการทำงานด้วย Block diagram ดังแสดงด้วยรูปที่ 1.1



รูปที่ 1.1 แสดง block diagram การทำงาน

จากรูปจะมีส่วนที่ทำหน้าที่ในการควบคุมหลักอยู่สองส่วนคือ ส่วนของคอมพิวเตอร์ส่วนบุคคลกับชุดควบคุมการหมุนของกล้องเพื่อติดตามการเคลื่อนไหวของวัตถุ โดยทั้งสองส่วนจะทำงานสัมพันธ์กัน

ภาคคอมพิวเตอร์ส่วนบุคคล: ในภาคนี้เป็นส่วนที่ใช้ในการประมวลผลสัญญาณวีดีโอโดยจะทำการเปรียบเทียบภาพเพื่อหาความแตกต่างของภาพระหว่างภาพอ้างอิงกับภาพที่ถ่ายในเวลาต่อมาและถ้าภาพทั้งสองมีความแตกต่างกันแสดงว่ามีการเคลื่อนที่ของวัตถุที่อยู่ภายในภาพดังนั้นก็จะทำการหาตำแหน่งของวัตถุพร้อมทั้งทำการแปลงระยะห่างระหว่างตำแหน่งของวัตถุกับตำแหน่งกึ่งกลางของภาพแล้วทำการแปลงเป็นจำนวนพัลส์ส่งออกทางพอร์ตอนุกรม RS 232 เพื่อไปควบคุมการการทำงานของภาคควบคุมภาคต่อไป

ภาคควบคุมการหมุนของกล้อง: ในภาคนี้จะประกอบด้วยบอร์ดควบคุมขนาดเล็กที่ใช้ IC MCS-51, บอร์ดขับกระแสสำหรับ stepping motor, ชุดเฟืองสำหรับหมุนกล้อง และตัวกล้องวีดีโอขนาดเล็ก การทำงานของภาคนี้สามารถอธิบายได้คร่าวๆ ดังนี้คือ เมื่อบอร์ควบคุมรับข้อมูลจากคอมพิวเตอร์ผ่านทางพอร์ตอนุกรม RS 232 แล้วจะนำข้อมูลนี้ไปทำการสร้างเป็นพัลส์เพื่อกำหนดตำแหน่งการหมุนของ stepping motor เพื่อให้กล้องทำการโฟกัสไปยังตำแหน่งของวัตถุและหลังจากที่หมุนไปยังตำแหน่งที่ต้องการแล้วชุดควบคุมก็จะทำการส่งข้อมูลกลับมาให้คอมพิวเตอร์เพื่อให้ทำการเปรียบเทียบภาพต่อไป

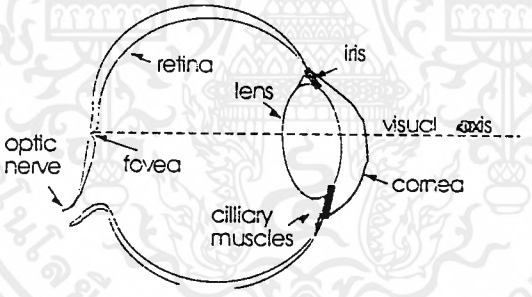
บทที่ 2
ทฤษฎีและหลักการ

2.1 ความเข้าใจเกี่ยวกับการมองเห็น (VISUAL PERCEPTION)

ในการที่จะทำการประมวลผลเกี่ยวกับภาพที่มองเห็นนั้น สิ่งสำคัญที่สุดก็คือวิธีการที่จะแปลงสิ่งที่มองเห็นให้มาอยู่ในรูปแบบของข้อมูลภาพ (image data) ที่สามารถนำมาประมวลผลได้

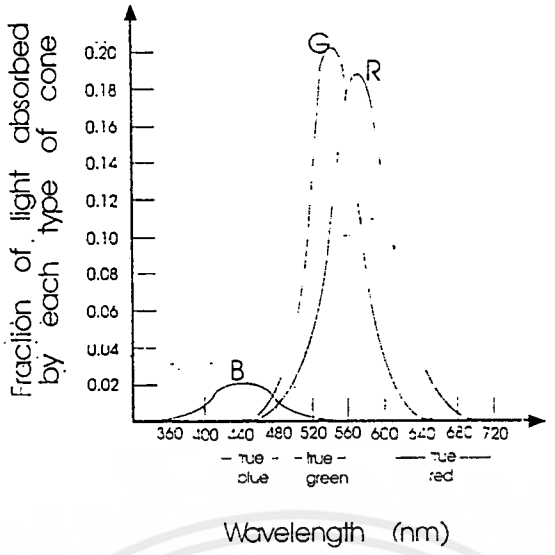
ข้อมูลภาพเป็นสิ่งที่แทนปริมาณทางกายภาพต่างๆ ดังเช่นค่า chromaticity และ luminance ค่า chromaticity นั้นเป็นคุณสมบัติของสีที่เกี่ยวกับแสงซึ่งถูกกำหนดโดยค่าความยาวคลื่น (wave length) ของมัน ส่วนค่า luminance นั้นจะแสดงคุณสมบัติของสีที่เกี่ยวกับปริมาณของแสง ซึ่งในการมองเห็นจริงๆ นั้น คุณสมบัติทางกายภาพที่กล่าวมาก็คือสีและค่าความสว่างของภาพนั่นเอง

ในการมองเห็นของมนุษย์นั้นเริ่มต้นจากการที่แสงที่เดินทางมาจากวัตถุแล้วหักเหผ่าน cornea (ส่วนที่เป็นเลนส์อยู่นอกสุดของดวงตา) ดังรูปที่ 2.1 ภายในดวงตาของคนเรานั้นจะมีกล้ามเนื้อที่มีชื่อว่า ciliary muscles ทำหน้าปรับเปลี่ยนรูปร่างของ cornea เพื่อทำการปรับระยะโฟกัสของภาพเพื่อให้ภาพตกกระทบที่เรตินา ซึ่งในเรตินาจะมีส่วนที่ทำหน้าที่จับแสง (photoreceptors)



รูปที่ 2.1 แสดงภาคตัดของตามนุษย์ในส่วนที่ทำหน้าที่เกี่ยวกับการมองเห็น

ตัวที่ทำหน้าที่ในการจับแสงมีอยู่สองชนิดด้วยกันก็คือ rods และ cones ซึ่งมีชื่อเรียกตามรูปร่างของมัน rods จะทำหน้าที่เกี่ยวกับการรับค่าความเข้มของแสง (intensity) ของภาพซึ่งไม่ได้ทำหน้าที่เกี่ยวกับการแยกรายละเอียดของสีของภาพ ซึ่งสิ่งที่ทำหน้าที่นี้ก็คือ cones ซึ่ง cones จะมีอยู่ด้วยกันสามประเภทซึ่งแยกตามค่าสีหลักสามสีคือ สีน้ำเงิน สีเขียว และสีแดง ซึ่ง cone แต่ละ ประเภทก็จะตอบสนองต่อสีนั้ นๆ ซึ่งสำหรับสีน้ำเงินมีค่าของการตอบสนองสูงสุด (peak response) อยู่ที่ 440 nm สีเขียวอยู่ที่ 545 nm และสีแดงอยู่ที่ 580 nm ดังรูปที่ 2.2



รูปที่ 2.2 การตอบสนองเชิงสเปคตรัมของ cones

อย่างไรก็ตาม ข่าวสารของภาพสี (color image information) ได้มีการจำแนกออกเป็น 3 อย่างด้วยกัน ได้แก่ hue saturation และ lightness ซึ่ง hue จะหมายถึงความแตกต่างระหว่างค่าของสี saturation หมายถึงค่าดีกรีของสีที่ซึ่งไม่ได้ถูกทำให้เจือจางโดยแสงสีขาว ค่า saturation นี้จะเป็นตัวลดค่ากลางของสี ในแสงสีขาวและสีบริสุทธิ์ (pure color) จะมีค่า saturation อยู่ในช่วงระหว่าง 0 ถึง 100 % lightness จะหมายถึงค่าความเข้มของแสงที่สะท้อนมาจากวัตถุ ซึ่งจะหมายถึงค่าของสีระหว่างสีขาวและสีดำ ซึ่งปกติเราจะเรียกค่าของสีที่อยู่ในช่วงนี้ว่าค่าระดับเทา (gray level)

นอกจากนี้ยังมีตัวแปรอีกตัวหนึ่งที่ชื่อ contrast หรือค่าความคมชัดของภาพ ซึ่งจะหมายถึงช่วงระหว่าง darkest region ของภาพกับ lightness region ของภาพ ซึ่งสามารถแสดงออกมาในรูปของสมการคณิตศาสตร์ได้ดังสมการที่ 2.1 ดังนี้

$$Contrast = \frac{I_{max} - I_{min}}{I_{max} + I_{min}} \qquad (2.1)$$

เมื่อ I_{max} : ค่าความเข้มที่มากที่สุดของภาพ

I_{min} : ค่าความเข้มที่น้อยที่สุดของภาพ

ภาพที่มีค่า contrast มากจะมีช่วงระหว่าง darkest region กับ lightnest region ที่กว้างมาก ซึ่งถ้าค่า contrast มาก เราจะสามารถมองเห็นรายละเอียดของภาพได้ดี

2.2 การแสดงแบบสี (COLOR REPRESENTATION)

โมเดลสี (color model, color space) เป็นแนวทางในการใช้แสดงเกี่ยวกับค่าสีและความสัมพันธ์ของสีต่าง ๆ ในการประมวลผลเกี่ยวกับภาพหลาย ๆ ระบบนั้นมีการใช้โมเดลสีที่แตกต่างกันตามเหตุผลต่างๆ ภาพสีที่ใช้ในอุตสาหกรรมสิ่งพิมพ์นั้นจะใช้โมเดลสี CMY ในระบบมอนิเตอร์สีที่ใช้หลอดภาพแบบ CRT หรือในคอมพิวเตอร์ทั่วๆ ไปนั้นจะใช้โมเดลสี RGBในระบบที่ต้องการประมวลผลเกี่ยวกับค่า hue, saturation และค่า intensity ที่แยกจากกันนั้นก็ต้องใช้โมเดลสี HSI

รูปที่ 2.3 แสดงความแตกต่างของแถบ luminance ในรูปของ actual intensity และ perceived intensity

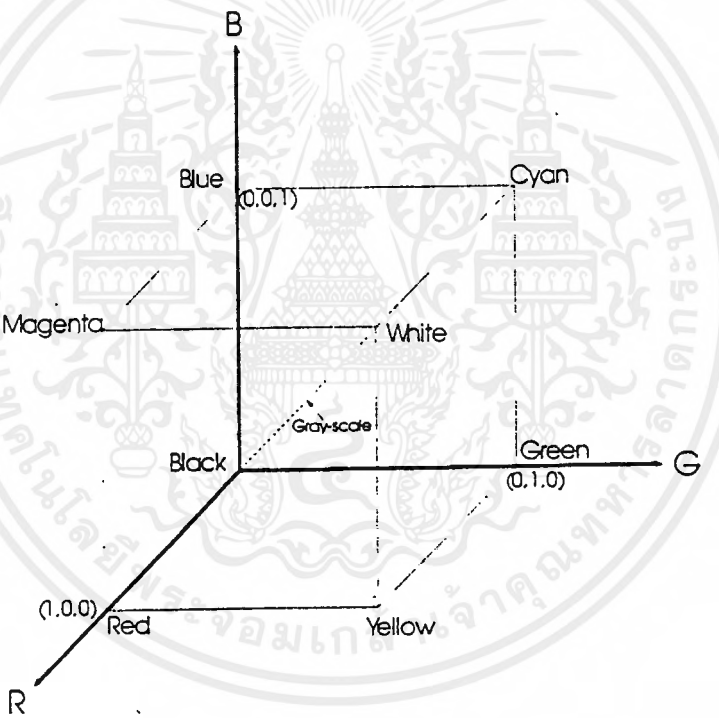
ในการสังเกตเห็นค่าสีของมนุษย์นั้นเป็นหน้าที่ของ cone ทั้งสามชนิด ดังนั้นโมเดลสีในระบบสี (color systems) จึงขึ้นอยู่กับค่าสีทั้งสาม ค่าสีเหล่านั้นจะถูกเรียกว่า trisimulus values

โมเดลสีต่างๆ เหล่านี้จะขึ้นอยู่กับค่า trisimulus values โมเดลสี YIQ ถูกใช้ในเรื่องระบบออกอากาศของสถานีโทรทัศน์ ในระบบพิกัด XYZ ไม่ได้เกี่ยวกับค่า trisimulus แต่ระบบพิกัดนี้จะใช้ในการอธิบายโมเดลสีต่าง ๆ เพื่อเป็นการง่ายต่อการอธิบายถึงค่าสีต่าง ๆ เราจะทำการนอมัลไลซ์ค่าสีต่างๆ ซึ่งจะทำให้ได้ค่าสีต่าง ๆ อยู่ในช่วง 0 จนถึง 1 (ดังเช่นในระบบสีแบบ 8 บิต จะทำการนอมัลไลซ์ด้วยค่า 255)

เนื่องจากในโครงงานนี้ใช้อินพุทจากกล้องพีซีวีดีโอ ซึ่งมีการจับภาพในรูปแบบของโมเดลสีแบบ RGB ดังนั้นในที่นี้จะขอกล่าวถึงเฉพาะโมเดลสี RGB เท่านั้น

2.2.1 โมเดลสีแบบ RGB

โมเดลสีชนิดนี้ประกอบด้วยการบวกกันของแม่สีหลักซึ่งได้แก่ แดง เขียว และน้ำเงิน ซึ่งค่าสีต่าง ๆ ในแถบสเปคตรัมของสีจะได้มาจากการผสมกันในอัตราส่วนที่แตกต่างกันของแม่สีทั้งสาม



รูปที่ 2.4 แสดงรูปลูกบาศก์สีของโมเดลแบบ RGB

โมเดลสี RGB นี้จะแสดงด้วยแกนของลูกบาศก์สามแกนในระนาบ 3 มิติ ซึ่งค่าสีแดง เขียว และน้ำเงินจะอยู่ที่มุมทั้งสามของแต่ละแกนดังแสดงด้วยรูปที่ 2.4 ซึ่งจะเห็นว่าค่าสีดำจะอยู่ที่จุดกำเนิด (origin) สีขาวจะอยู่ที่มุมตรงข้ามกับสีดำ ค่าของสีในช่วงระดับเทาจะอยู่ตามเส้นที่เชื่อมระหว่างค่าสีดำและค่าสีขาว จากรูปถ้าเป็นในระบบการแสดงสีแบบ 24 บิต (แบ่งออกเป็น 8 บิต ต่อแม่สีหนึ่งสี) นั้น ค่าสีแดงจะถูกแทนด้วยค่า (255,0,0)

โมเดลสี RGB ง่ายต่อการออกแบบและใช้งานในระบบคอมพิวเตอร์กราฟฟิค แต่ไม่เหมาะสำหรับการประยุกต์ในงานอื่น ๆ ค่าของสีแดง สีเขียว และสีน้ำเงินจะมีความสัมพันธ์กันอย่างมากซึ่งจะเป็นการยากที่จะนำไปประมวลผลเกี่ยวกับภาพ ดังเช่นการทำฮีสโตแกรม อีควอไลเซชัน (histogram equalization) หรือการประมวลผลที่เกี่ยวกับค่า intensity เท่านั้น ซึ่งในกรณีนี้จะเป็นการง่ายถ้าทำการประมวลผลในรูปแบบของโมเดลสีแบบ HSI

หลายครั้งที่มีความจำเป็นที่จะต้องทำการแปลงภาพจากโมเดลสี RGB ให้อยู่ในรูปแบบของภาพแบบระดับเทา (gray scale image) เพื่อความสะดวกในการประมวลผล ซึ่งในโครงงานชิ้นนี้ก็เช่นกัน

ในการแปลงภาพจากโมเดลสี RGB ให้อยู่ในรูปแบบภาพระดับเทานั้น สามารถทำได้โดยการใช้สมการการแปลงดังแสดงในสมการที่2.2 ดังนี้

$$\text{Gray scale intensity} = 0.299R+0.587G+0.114B \qquad (2.2)$$

ซึ่งในสมการนี้ใช้สำหรับการแปลงภาพจากมาตราฐาน NTSC หรืออีกสมการหนึ่งที่ใช้ในการแปลงซึ่งเป็นสมการแบบง่ายดังแสดงในสมการที่ 2.3 ดังนี้

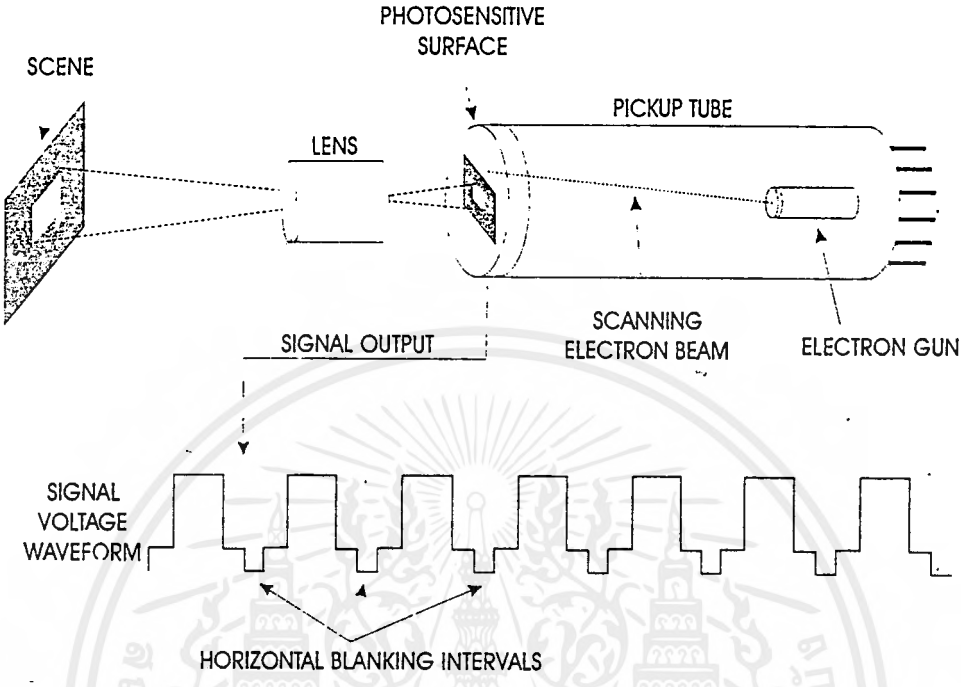
$$\text{Gray scale intensity} = 0.333R+0.333G+0.333B \qquad (2.3)$$

ซึ่งในสมการแบบนี้จะถูกใช้ในการแปลงจาก RGB ไปเป็น HSI แต่ในโครงงานชิ้นนี้จะใช้สมการการแปลงในสมการที่ 2.2

2.3 กล้องวีดีโอ (VIDEO CAMERAS)

การจับภาพวีดีโอทางอิเล็กทรอนิกส์นั้นสามารถทำได้โดยการใช้อุปกรณ์จำพวกกล้องวีดีโอ ดังแสดงในรูปที่ 2.5 ซึ่งแสดงให้เห็นถึงกล้องวีดีโอประเภทขาว - ดำ (monochrome) ซึ่งประกอบด้วยเลนส์ซึ่งมีหน้าที่ในการโฟกัสให้ภาพไปตกกระทบที่พื้นผิวของอุปกรณ์ที่ทำหน้าที่รับแสง (photosensitive) ของกล้องหรือส่วนที่เป็นโซลิดสเตทเซนเซอร์ (solid state sensor) หรือบางทีเรียกอุปกรณ์ชนิดนี้ว่า charge couple device (CCD) ตัวเซนเซอร์นี้จะทำหน้าที่ในการแปลงแสงที่สะท้อนออกมาจากภาพให้อยู่ในรูปของประจุไฟฟ้า ซึ่งจะอยู่ในรูปแบบของภาพทางอิเล็กทรอนิคส์ซึ่งคล้ายกับภาพที่เกิดจากแสงเว้นแต่แต่ละจุดของภาพจะเป็นปริมาณของประจุไฟฟ้าที่มีสัดส่วนโดยตรงกับค่าแสงที่มาตกกระทบที่แต่ละจุด ในรูปแบบของคอมพิวเตอร์นั้น ภาพที่เกิดจากประจุไฟฟ้าจะเป็นการกระทำที่พร้อมกัน (parallel operation) เพราะประจุที่ทุกๆ จุดนั้นจะปรากฏอยู่ในเวลาเดียวกัน รูปแบบของประจุไฟฟ้าถูกอ่านออกมาเป็นลำดับอนุกรมโดยการสแกนด้วยลำอิเล็กตรอนในหลอด vacuum-tube sensor หรือทางอิเล็กทรอนิกส์ใน CCD เมื่อมีการสแกนผ่านแต่ละจุดบนพื้นผิว ประจุไฟฟ้าที่จุดนั้นๆ จะถูกแปลงให้อยู่ในรูปแบบของค่าศักดาไฟฟ้าส่งออกทางสายสัญญาณ รูปที่ 2.5 แสดงถึงสัญญาณที่เป็นผลมาจากการสแกนบริเวณภาพของวัตถุที่มีลักษณะเป็นสี่เหลี่ยมมุมฉากสีขาวที่อยู่บนพื้นสีเทา สังเกตว่าสัญญาณจะถูกอินเตอร์รัพต์ในแต่ละช่วงเวลาที่มีการวกกลับของการสแกน ซึ่งการทำการอินเตอร์รัพต์นี้จะถูกเรียกว่า

blanking intervals ซึ่งจะนำไปสู่การทำรูปแบบของสัญญาณวีดีโอที่ง่ายขึ้นเพื่อใช้ในการประมวลผลสัญญาณวีดีโอ



รูปที่ 2.5 กล้องวีดีโอประเภทขาว-ดำและสัญญาณเอาท์พุทที่ได้จากกล้องวีดีโอ

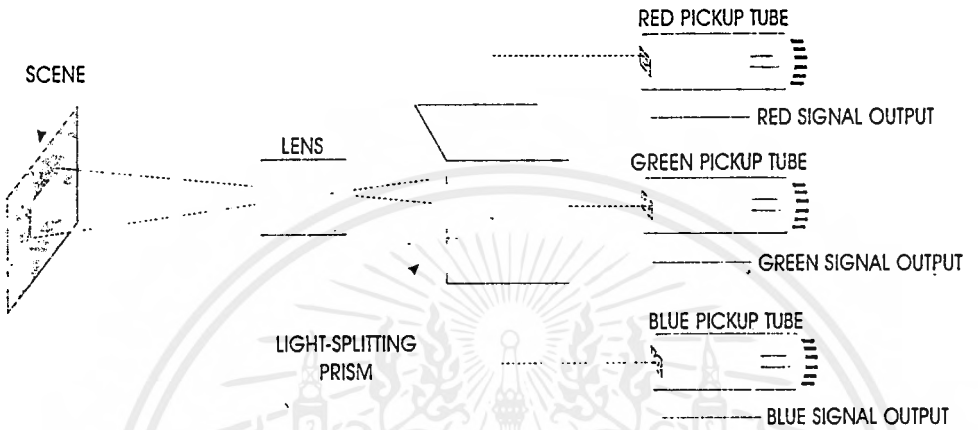
ในสัญญาณวีดีโอจะมี blanking intervals อยู่สองชนิดด้วยกันคือแบบแนวตั้ง (horizontal) และแบบแนวนอน (vertical) horizontal blanking intervals นั้นปรากฏอยู่ในทุกๆ เส้นที่มีการวกกลับของการสแกนตามแนวนอน และ vertical blanking intervals นั้นปรากฏอยู่ในทุกๆ ครั้งที่มีการวกกลับของการสแกนตามแนวตั้ง (ทุกๆ ครั้งที่มีการเปลี่ยนเฟรม)

2.3.1 ระบบวีดีโอสี (COLOR VIDEO SYSTEMS)

จากการที่เราได้กล่าวถึงระบบวีดีโอแบบ ขาว-ดำ มานั้น ในปัจจุบันได้มีการผลิตระบบวีดีโอแบบสีขึ้นมาโดยใช้ทฤษฎี tristimulus theory ซึ่งมีพื้นฐานมาจากความจริงที่ว่าในตาของมนุษย์เรานั้นมีสิ่งที่ทำหน้าที่ในการจำแนกสีจากการมองเห็นอยู่สามชนิดและจากการที่นำสีเหล่านั้นทั้งสามสีมาผสมกันทำให้เราสามารถมองเห็นสีได้ทุกสี จากที่กล่าวมานั้นสีใด ๆ ก็ตามสามารถเกิดได้จากการผสมกันของแม่สีหลักสามสี (three primary colors) ซึ่งแม่สีในระบบวีดีโอนั้นก็ได้แก่สีแดง, สีเขียว และสีน้ำเงิน แสงของสีเหล่านี้สามารถนำมาผสมกันเพื่อที่จะทำการสร้างสีใดๆ ก็ได้

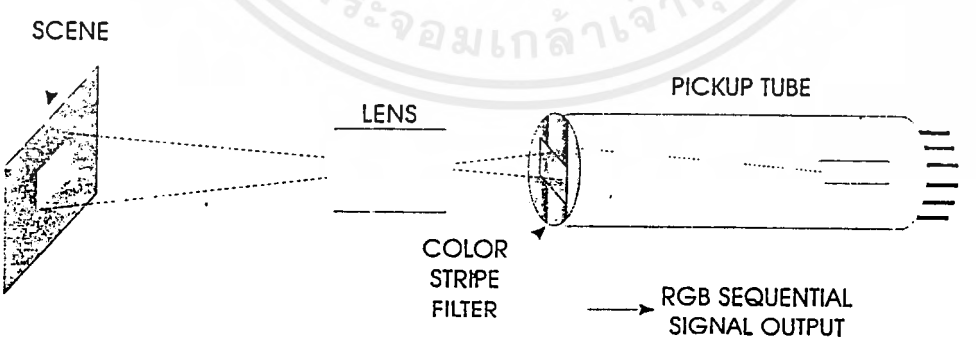
ในการสร้างระบบวีดีโอแบบสีนั้นสามารถทำได้โดยการใช้กล้องวีดีโอสามตัวจับไปที่ภาพเดียวกันโดยการใช้ตัวกรองสี (color filter) ที่เหมาะสมในส่วนหน้าของกล้องแต่ละตัวจะทำให้กล้องวีดีโอแต่ละตัวนั้น

สามารถมองเห็นภาพเพียงสีของแม่สีเพียงสีเดียวเท่านั้น จากการที่ทำให้กล้องวีดีโอทั้งสามตัวสามารถจับภาพภาพเดียวกันอย่างถูกต้อง (กล้องวีดีโอทั้งสามตัวต้องจับภาพๆ เดียวกันในช่วงเวลาหนึ่งๆ) ได้นั้นเราจะใช้อุปกรณ์ที่เรียกว่า light-splitting device ซึ่งมีลักษณะเหมือนกับปริซึม ดังแสดงในรูปที่ 2.6 ซึ่งเราจะเรียกอุปกรณ์ตัวนี้ว่า three-sensor RGB camera



รูปที่ 2.6 กล้องวีดีโอแบบสีสามเซนเซอร์ (three-sensor color camera)

สำหรับในการแสดงผลนั้น สัญญาณเอาท์พุททั้งสามสัญญาณจากกล้องวีดีโอจะถูกต่อเข้ากับตัวแสดงสีซึ่งมีอินพุทเป็นค่า RGB จอมอนิเตอร์สำหรับแสดงภาพสีนั้นจะใช้ CRT แบบสีซึ่งมีลำบีมสำหรับการสแกนสามลำบีม (แต่ละบีมสำหรับแม่สีแต่ละสี)



รูปที่ 2.7 กล้องวีดีโอแบบสีเซนเซอร์เดี่ยว (A single-sensor color camera)

โดยหน้าจอภาพนั้นจะถูกออกแบบให้แต่ละบีมตกกระทบลงบน phosphor pattern ของสีที่เหมาะสม phosphor pattern จะมีลักษณะเป็นอาร์เรย์ (array) ของจุด (dots) หรือเส้นเล็กๆ พอที่จะมองไม่เห็นถึงเม็ดสีแต่

ละเม็ดสี ซึ่งวิธีนี้ใช้แทนการผสมกันของแม่สีทั้งสามเพื่อทำให้เกิดค่าสีที่ต้องการซึ่งในระบบการแสดงผลของคอมพิวเตอร์และโทรทัศน์ก็ใช้วิธีการแบบนี้

ในทำนองเดียวกันกับที่กล่าวมาในการแสดงผลแบบ three-dot สามารถทำได้โดยใช้ตัวเซนเซอร์เพียงตัวเดียวในการจับภาพสี รูปแบบ (pattern) ของตัวกรองสีแบบจุดหรือแบบเส้นถูกวางบนพื้นผิวสำหรับจับภาพของตัวเซนเซอร์ ดังแสดงด้วยรูปที่ 2.7 ขณะที่ตัวเซนเซอร์ถูกสแกนนั้น สัญญาณวีดีโอจะเป็นในลักษณะของลำดับของแม่สีทั้งสาม สัญญาณนี้จะถูกประมวลผลทางไฟฟ้าโดยการแยกลำดับของแม่สีเข้าไปในสัญญาณ RGB บนสายสัญญาณทั้งสามสาย

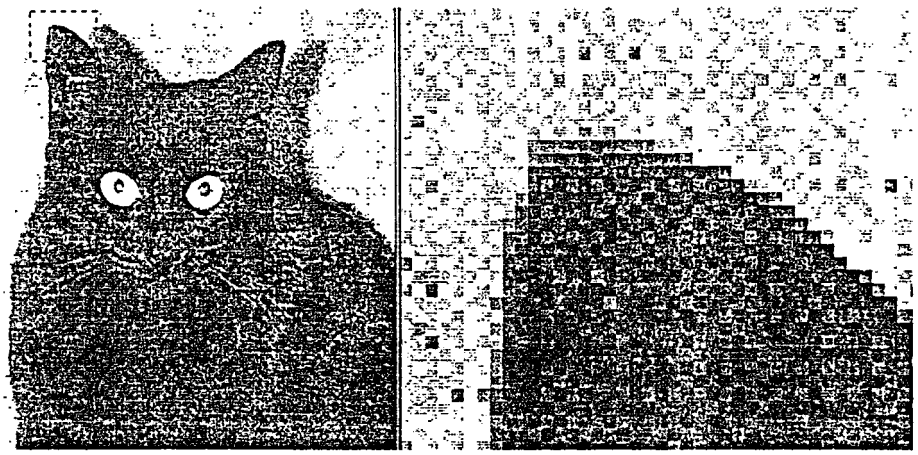
แต่จากการใช้สายสัญญาณสามสายในการส่งสัญญาณ RGB นั้นไม่มีความเหมาะสมในการนำไปใช้กับระบบออกอากาศของสถานีโทรทัศน์ซึ่งมีเพียงหนึ่งช่องสัญญาณ ดังนั้นจึงได้มีการเข้ารหัส (encoding) โดยการรวมสัญญาณทั้งสามเข้าสู่สัญญาณเดียวที่เรียกว่า composite signal เพื่อใช้ในการออกอากาศ ซึ่งในสหรัฐอเมริกาเรียกการเข้ารหัสแบบนี้ว่า NTSC encoding ซึ่งย่อมาจาก National Television Systems Commitee ซึ่งเป็นองค์กรที่กำหนดมาตราฐานสำหรับโทรทัศน์สีในสหรัฐอเมริกาในปี คศ. 1953 นอกจากนี้ยังมีวิธีในการเข้ารหัสแบบอื่นๆ อีกเช่น PAL และ SECAM

2.3.2 ดิจิตอลวีดีโอ (DIGITAL VIDEO)

สัญญาณอานาล็อควีดีโอ (monochrome) มีความคล้ายคลึงกับสัญญาณเสียง เว้นแต่โครงสร้างของสัญญาณจะประกอบไปด้วยจำนวนเส้นและเฟรมภาพที่เกิดจากการสแกน ขบวนการในการสร้างสัญญาณวีดีโอนั้นก็มีความคล้ายกับในกรณีของการสร้างสัญญาณเสียงซึ่งประกอบด้วยขบวนการในการ sampling และ quantizing อย่างไรก็ตามอัตราการแซมปลิ้งที่ใช้นั้นจะเป็นตัวกำหนดจำนวนของตัวอย่างในการสุ่มค่าของแต่ละเส้นของการสแกนแต่ละครั้ง ซึ่งตัวอย่างของสัญญาณที่ได้จากการสุ่มค่านี้จะเป็นตัวแทนของจุดแต่ละจุดบนภาพ ซึ่งเรียกจุดเหล่านี้ว่าพิกเซล (pixel) หรือ ส่วนประกอบของภาพ (picture element) รูปที่ 2.8 แสดงถึงการขยายภาพให้ใหญ่ขึ้นเพื่อที่จะสามารถมองเห็นพิกเซลแต่ละพิกเซลได้

ดังตัวอย่าง ถ้าเราต้องการจำนวนพิกเซลในแต่ละเส้นของการสแกนจำนวน 640 พิกเซล และถ้าเราใช้อัตราการสแกนตามมาตราฐานของโทรทัศน์ ทั่วๆ ไป นั่นหมายความว่าความถี่ที่ใช้ในการสุ่มตัวอย่างสัญญาณแต่ละครั้งจะประมาณ 12.3 MHz ซึ่งคำนวนจากการคูณความถี่ที่ใช้ในการสแกนแต่ละเส้น (15,734 Hz) ด้วย 640 และนำไปหารช่วงคาบเวลาในการสแกนแต่ละเส้นซึ่งมีค่าเท่ากับ 0.82

ในระบบดิจิตอลวีดีโอนั้น ค่ารีโซลูชั่นสามารถคำนวณได้จากจำนวนของพิกเซลในแนวตั้งและแนวนอน ซึ่งจะต้องระบุด้วยค่าทั้งสองดังเช่นตัวอย่าง ในระบบการแสดงผลแบบ VGA ที่มีการแสดงผลแบบ 640× 480 พิกเซล นั่นหมายความว่ามีค่ารีโซลูชั่นเท่ากับ 640 ในแนวนอนและ 480 ในแนวตั้ง สังเกตว่าความสูงของภาพที่คิดจากจำนวนพิกเซลและจำนวนเส้นของภาพนั้นจะไม่ได้วัดในหน่วยเดียวกัน



รูปที่ 2.8 แสดงภาพปกติทางซ้ายและพิกเซลของภาพบริเวณเส้นประซึ่งถูกขยายให้ใหญ่ขึ้นทางขวา

ในระบบดิจิตอลออดิโอนั้น เรากล่าวถึงจำนวนบิตที่ใช้ในการสุ่มตัวอย่างสัญญาณแต่ละครั้งนั้นเป็นตัววัดคุณภาพของเสียงได้ ซึ่งในระบบดิจิตอลวีดีโอเทียบได้กับจำนวนบิตต่อพิกเซล (bit per pixel : bpp)

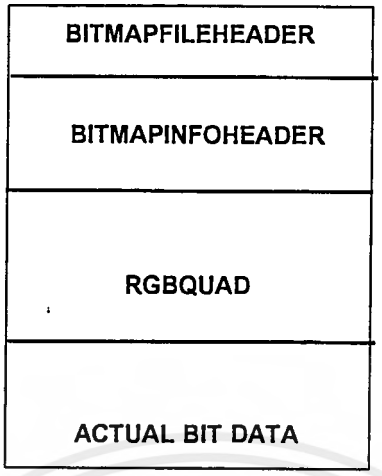
เนื่องจากในการสุ่มตัวอย่างสัญญาณแต่ละครั้งนั้นจะได้เป็นหนึ่งพิกเซล ในระบบ monochrome ที่มีคุณภาพสูงนั้นจะใช้ 8 bpp ซึ่งหมายถึงจำนวนระดับเทา (gray level) ของภาพจะเท่ากับ 256 ระดับ ในระบบการแสดงผลแบบสีนั้นเราต้องการหนึ่งช่องสัญญาณแบบ monochrome ต่อแม่สีแต่ละสี (แดง,เขียว และน้ำเงิน) ซึ่งจะต้องใช้จำนวนบิตทั้งหมดเป็น 24 บิต ซึ่งจะได้ค่าระดับสีที่เป็นไปได้คือ 16,777,216 สี ในระบบ digital video บางระบบจะใช้ 24 บิตต่อพิกเซล แต่ส่วนใหญ่จะใช้จำนวนบิตต่อพิกเซลที่น้อยกว่านี้ ดังเช่น 8 bpp หรือ 16 bpp อัตราข้อมูลในระบบวีดีโอนั้นเราสามารถคำนวนได้จากสมการที่ 2.4 ดังนี้

$$\textit{Data rate} = \frac{(SamplingFrequency) \times (bpp)}{8} \qquad (2.4)$$

2.4 ไฟล์ DIB (DIB FILE)

DIB ไฟล์ (device independent bitmap file) เป็นไฟล์ภาพประเภท bitmap ซึ่งไม่ขึ้นกับอุปกรณ์ที่สร้างไฟล์นี้ขึ้นมา ปกติมีอยู่ 2 ชนิดด้วยกันคือ ชนิดแรกใช้ในระบบปฏิบัติการ window และ อีกชนิดใช้ในระบบปฏิบัติการ OS/2 Presentation Manager แต่ในโครงงานนี้จะกล่าวถึงเฉพาะแบบแรกเท่านั้น

ใน Windows DIB นั้น รูปแบบของไฟล์จะแสดงได้ดังรูปที่ 2.9 ซึ่งประกอบด้วย 4 ส่วนด้วยกัน ส่วนแรกจะเรียกว่า BITMAPFIL-EHEADER, ส่วนที่สองคือ BITMAPINFOHEADER, ส่วนที่สามคือ RGBQUAD และส่วนสุดท้ายจะเป็น ACTUAL BIT DATA



รูปที่ 2.9 โครงสร้างของไฟล์ DIB บนระบบปฏิบัติการวินโดวส์

จากรูปส่วนแรกจะเป็น BITMAPFILEHEADER จะประกอบด้วยฟิลด์ (fields) ย่อยๆ 5 ฟิลด์ดังแสดงดังตารางที่ 2.1

ตารางที่ 2.1 ฟิลด์ต่างๆภายใน BITMAPFILEHEADER

Field	Size	Description
bfType	UINT	ไบต์แสดง BM มีค่า 4D42
bfSize	DWORD	ขนาดของไฟล์
bfReserves1	UINT	0
bfReserves2	UINT	0
bfOffBits	DWORD	ค่าออฟเซ็ทของ data bit จากจุดเริ่มของไฟล์

ส่วนที่สองจะเป็นส่วน BITMAPINFOHEADER ซึ่งเป็นส่วนที่เกี่ยวข้องกับข้อมูลต่างๆ ของไฟล์ ดังตารางที่ 2.2

ตารางที่ 2.2 โครงสร้างของ BITMAPINFOHEADER

Field	Size	Description
biSize	DWORD	ขนาดของ BITMAPINFOHEADER
biWidth	LONG	ความกว้างของภาพในหน่วย pixels
biHeight	LONG	ความสูงของภาพในหน่วย pixels
biPlanes	WORD	จำนวน color planes เซ็ทเป็น 1
biBitCount	WORD	จำนวน bit สี ต่อ 1 pixel
biCompression	DWORD	แสดงการบีบอัด (0 แสดงว่าไม่บีบ)
biSizeImage	DWORD	ขนาดภาพในหน่วยไบต์ (ใช้เฉพาะกรณี ค่า biCompression ไม่เป็น 0)
biXPelsPerMeter	LONG	ค่า resolution ตามแนวนอนในหน่วย pixel ต่อ เมตร
biYPelsPerMeter	LONG	ค่า resolution ตามแนวตั้งในหน่วย pixel ต่อ เมตร
biClrUsed	DWORD	จำนวนสีที่ใช้ในภาพ
biClrImportant	DWORD	จำนวนสีที่มีความสำคัญที่ถูกใช้ในภาพ

ส่วนที่สามจะเป็นส่วน RGBQUAD ซึ่งเป็นส่วนที่เกี่ยวข้องกับตารางสี (palette) ถ้าค่า biBitCount มีค่าเป็น 1 นั่นหมายถึงจะมี RGBQUAD อยู่ 2 ตารางสี ถ้าค่า biBitCount มีค่าเป็น 4 จะมี RGBQUAD อยู่ 16 ตารางสี และถ้าค่า biBitCount มีค่าเป็น 8 จะมี RGBQUAD อยู่ 256 ตารางสี แต่ถ้า biBitCount มีค่าเป็น 24 จะเป็นกรณีพิเศษซึ่งจะไม่มีตารางสีโดยค่า 24 บิต นั้นจะเป็นค่าสีของ RGB โดยตรงโดยแบ่งออกเป็นสีละ 8 บิต ตารางที่ 2.3 แสดงถึงค่าของ fields ต่างๆ ที่อยู่ในส่วน RGBQUAD

ตารางที่ 2.3 โครงสร้างของ RGBQUAD

Field	Size	Description
rgbBlue	BYTE	ค่าความเข้มของสีน้ำเงิน
rgbGreen	BYTE	ค่าความเข้มของสีเขียว
rgbRed	BYTE	ค่าความเข้มของสีแดง
rgbReserved	BYTE	เซ็ทเป็นค่า 0

2.5 การสื่อสารข้อมูลแบบอนุกรมตามมาตรฐาน RS-232-C

2.5.1 ลักษณะทั่วไป

เนื่องจากความต้องการในการสื่อสารข้อมูลผ่านทางเครือข่ายโทรศัพท์มีมากขึ้นเรื่อยๆ ดังนั้นจึงต้องมีการกำหนดมาตรฐานที่เรียกว่า RS-232-C ขึ้นเพื่อใช้เป็นมาตรฐานจากอุปกรณ์ที่ถูกผลิตจากบริษัทต่างๆ ในสหรัฐอเมริกา Bell System Operating Telephone companies เป็นบริษัทหลักบริษัทแรกที่เป็นผู้ผลิตและติดตั้งระบบสื่อสารข้อมูลและเป็นผู้ผลิตอุปกรณ์ต่างๆ ที่ใช้ในการอินเตอร์เฟสอุปกรณ์ดิจิตอลกับเครือข่ายโทรศัพท์ระยะใหญ่ อุปกรณ์นี้ก็คือ Bell Modem ซึ่งถูกพัฒนาโดย Bell Laboratory และถูกใช้เป็นมาตรฐานในงานอุตสาหกรรมจนถึงปัจจุบันนี้ ขณะที่อุปกรณ์เกี่ยวกับคอมพิวเตอร์ต่างๆ ได้ถูกพัฒนาขึ้นเรื่อยๆ โดยบริษัทต่างๆ เทอร์มินัลและอุปกรณ์อื่นๆ มักถูกออกแบบให้สามารถอินเตอร์เฟสกับ Bell Modem ได้ ดังนั้นความต้องการข้อมูลเกี่ยวกับข้อกำหนดในการอินเตอร์เฟสกับโมเด็มจึงมีมากขึ้นเรื่อยๆ เพื่อตอบสนองต่อความต้องการนี้ EIA ,Bell system และผู้ผลิตโมเด็มรายอื่นๆ จึงร่วมมือกันตั้งมาตรฐาน RS-232-C ขึ้น

มาตรฐาน RS-232-C ได้ถูกพิมพ์โดย EIA ในปี ค.ศ. 1969 ตังอักษร RS แทน “Recommended Standard”, 232 แทนหมายเลขของมาตรฐาน ส่วนอักษร C แสดงให้รู้ว่าได้รับการแก้ไขกี่ครั้ง

ตามมาตรฐาน RS-232-C ที่ถูกตีพิมพ์โดย EIA ได้กล่าวถึงการสื่อสารข้อมูลระหว่าง Data Terminal Equipment (DTE) และ Data Communication Equipment (DCE) (ในปัจจุบันตัวย่อ DCE จะแทน Data Circuit Terminating Equipment) โดยคำจำกัดความของ DTE และ DCE แสดงได้ดังนี้

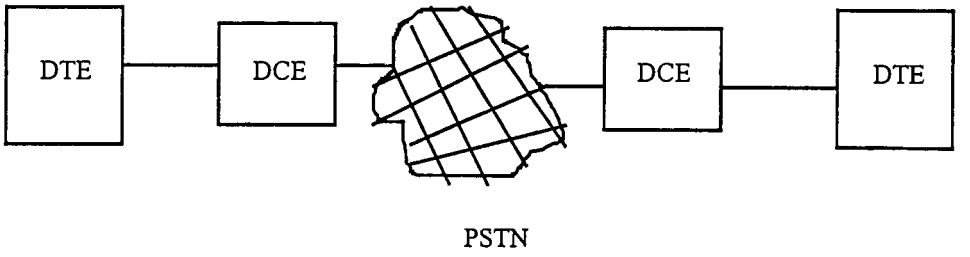
DTE : เป็นอุปกรณ์ซึ่งมีลักษณะดังนี้

- เป็นอุปกรณ์ ที่ประกอบไปด้วยตัวส่งข้อมูล (data source) หรือตัวรับข้อมูล (data sink) หรือเป็นทั้งตัวส่งและตัวรับข้อมูลก็ได้

- เป็นอุปกรณ์ที่ประกอบด้วย function unit ต่อไปนี้ control logic buffer store และอุปกรณ์อินพุทหรือเอาท์พุทจำนวนหนึ่งตัวหรือมากกว่าก็ได้ หรือรวมเครื่องคอมพิวเตอร์เข้าไปด้วยก็ได้ DTE อาจจะรวมส่วน error control , synchronization และความสามารถในการบ่งหรือระบุว่าต้องการเกี่ยวข้องกับอุปกรณ์ใด (station identification capability) เข้าไปด้วยก็ได้

DCE: เป็นอุปกรณ์ที่มีฟังก์ชั่นการทำงานต่างๆ ที่ทำให้เกิดการเชื่อมต่อ , ทำให้การเชื่อมต่อยังดำเนินต่อไป รวมถึงยุติการเชื่อมต่อ นอกจากนี้ยังใช้เปลี่ยนลักษณะของสัญญาณและสร้างรหัสสัญญาณต่างๆ ที่จำเป็นต้องใช้ในการสื่อสารข้อมูลระหว่าง DTE และ DCE โดย DCE อาจเป็นส่วนใดส่วนหนึ่งของคอมพิวเตอร์หรือไม่ก็ได้

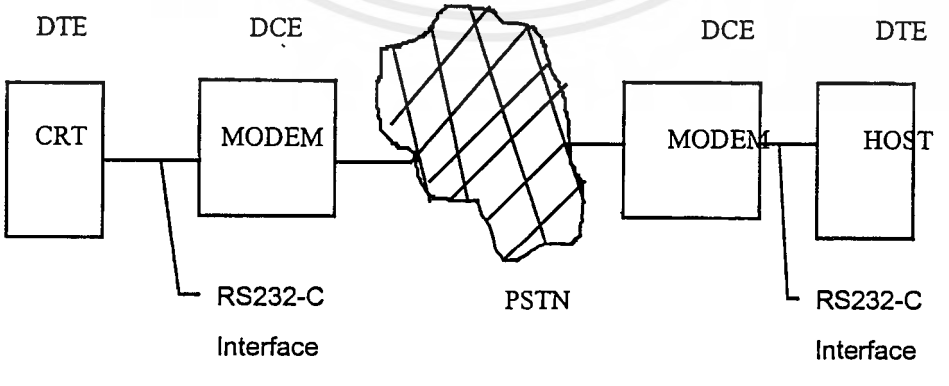
ลักษณะของ DTE และ DCEที่ใช้ในการสื่อสารข้อมูลได้แสดงไว้ในรูปที่ 2.10 ตามลักษณะการทำงานที่ได้อธิบายไว้ข้างต้น



รูปที่ 2.10 ลักษณะของ DCE และ DTE ที่ใช้ในการสื่อสารข้อมูล

ในความเป็นจริงแล้ว DTE มักจะแทนแหล่งกำเนิดข้อมูลแหล่งแรก และ/หรือ อุปกรณ์ที่เป็นแหล่งรับข้อมูลแหล่งสุดท้าย (ดังแสดงในรูปที่ 2.10) เช่นเครื่องพิมพ์ (เป็นอุปกรณ์ที่รับข้อมูลได้อย่างเดียว) จะเป็น DTE เพราะเป็นอุปกรณ์ที่รับข้อมูลเป็นตัวสุดท้าย ส่วน DCE เป็นอุปกรณ์ที่ทำให้การสื่อสารข้อมูลระหว่างแหล่งกำเนิดกับตัวรับข้อมูลที่ปลายทางทำได้สะดวกขึ้น ตัวอย่างหนึ่งของ DCE ก็คือโมเด็ม เพื่อความเข้าใจเพิ่มขึ้นขอให้ดูตัวอย่างต่อไปนี้ประกอบ

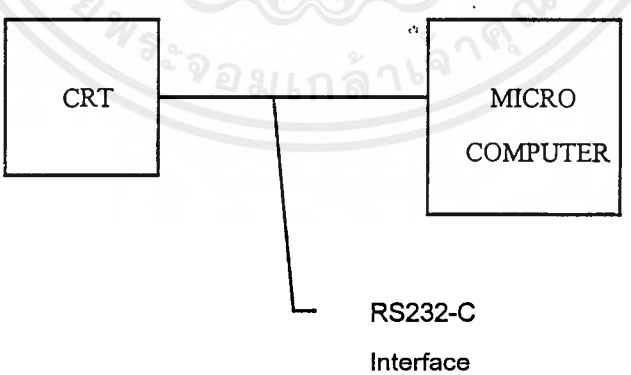
จากรูปที่ 2.11 CRT ถูกต่อเข้ากับคอมพิวเตอร์ปลายทางหลัก (remote host computer) ซึ่งใช้เป็นระบบจัดสรรเวลา (time sharing service) โดยการเชื่อมต่อผ่านสายโทรศัพท์ จากรูปที่ 2.11 นี้จะเห็นได้ว่ามี DCE และ DTE อยู่อย่างละ 2 ตัว DTE ตัวแรกเป็นเทอร์มินัล CRT ส่วน DTE อีกตัวหนึ่งเป็นคอมพิวเตอร์ปลายทางหลัก เหตุที่ทั้ง 2 ตัวเป็น DTE ก็เพราะคอมพิวเตอร์ปลายทางหลักแทนตัวส่งข้อมูล (แหล่งข้อมูล) แหล่งแรก ส่วน CRT เป็นตัวรับข้อมูลตัวสุดท้าย ส่วนโมเด็มทั้ง 2 ตัวที่ต่อที่ปลายสายโทรศัพท์ทั้ง 2 ด้านเป็น DCE เพราะโมเด็มทำหน้าที่เปลี่ยนลักษณะสัญญาณซึ่งทำให้สามารถทำการสื่อสารข้อมูลผ่านทางเครือข่ายโทรศัพท์ได้ ส่วนวงจรติดต่อ (circuit) ที่ใช้ในการสื่อสารข้อมูลระหว่าง DCE ทั้งสองด้านนั้นเราใช้การเชื่อมโยงทางโทรศัพท์ (telephone link) ส่วนการเชื่อมโยงระหว่าง DCE และ DTE นั้นใช้ RS-232-C เป็นมาตรฐานหลัก



รูปที่ 2.11 ลักษณะระบบที่ใช้ในตัวอย่างที่ 1

สำหรับเครือข่ายโทรศัพท์ที่ทำให้เกิดการเชื่อมต่อระหว่าง DCE ทั้ง 2 ตัวนั้นเราจะไม่พูดถึง (เพราะว่ามันเป็นระบบมาตรฐานอยู่แล้ว) ดังนั้นเราจะถือว่าเครือข่ายโทรศัพท์เป็น black box เราไม่จำเป็นต้องเข้าไปยุ่งเกี่ยวกับการเดินทางของข้อมูลในเครือข่ายโทรศัพท์ แต่อย่างไรก็ตามข้อสำคัญประการหนึ่งที่ควรสนใจก็คือคุณสมบัติของ black box นี้ ตัวอย่างเช่น ค่าหน่วงเวลา (delay) ที่เกิดขึ้นในการส่งข้อมูลผ่านเครือข่ายโทรศัพท์เทียบกับค่าที่เกิดขึ้นในการส่งข้อมูลโดยใช้ดาวเทียมนั้นมีค่าแตกต่างกันมาก การเดินทางของข้อมูลผ่านทางสายโทรศัพท์จะใช้เวลาประมาณ 1 มิลลิวินาทีต่อระยะทาง 100 ไมล์โดยประมาณ (แต่อย่างไรก็ตามการเชื่อมต่อของสายโทรศัพท์ผ่านทางชุมสายต่างๆ เป็นระยะทางไกลๆ สามารถทำให้ค่าหน่วงเวลานี้มีค่าสูงมากถึง 20 มิลลิวินาทีได้) ส่วนในการส่งข้อมูลผ่านดาวเทียมนั้นระยะทางจากพื้นโลกถึงดาวเทียมมีระยะทางเท่ากับ 22,000 ไมล์ แต่ละ big hop (big hop คือการเชื่อมโยงโดยดาวเทียม: satellite link) ที่ใช้ในการส่งข้อมูลนั้นกินเวลาอีก 700 มิลลิวินาที (จากพื้นโลกไปดาวเทียมแล้วกลับมายังพื้นโลกอีก) สำหรับทุกครั้งที่ใช้ในการส่งจากจุดหนึ่งไปยังอีกจุดหนึ่ง เวลาที่เพิ่มขึ้นนี้จะเป็นตัวตัดสินว่าเราควรจะเลือกใช้ตัวกลางที่ใช้ในการสื่อสารข้อมูลตัวไหนจึงจะเหมาะสม จากช่วงเวลาที่เสียไปตามที่กล่าวไว้ข้างต้นจะเห็นได้ว่าถ้าคอมพิวเตอร์ปลายทางหลักสะท้อน (echo) อักขระแต่ละตัวที่ได้รับจากคอมพิวเตอร์ต้นทางกลับมาที่คอมพิวเตอร์ต้นทางอีกครั้งหนึ่งจะกินเวลา(เดินทางไปกลับ)ถึง 1.4 วินาที (1400 มิลลิวินาที) ซึ่งค่าหน่วงเวลานี้มีค่ามากเกินกว่าที่จะนำไปใช้งานได้

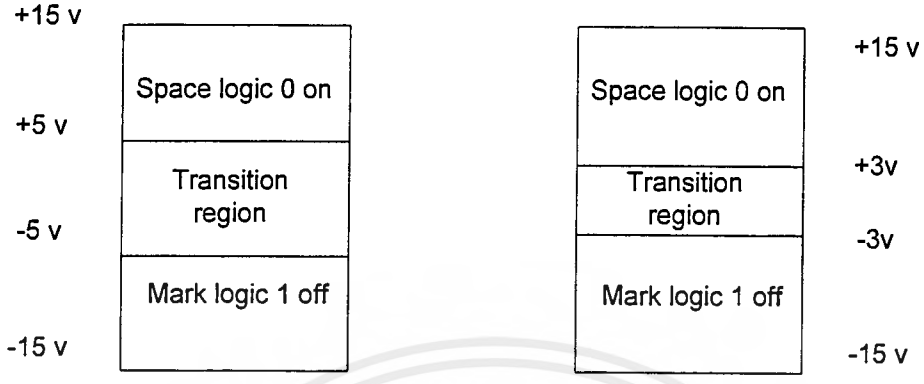
สำหรับอีกตัวอย่างหนึ่งนั้นสามารถพิจารณาได้จากรูปที่ 2.12 โดยจะเห็นได้ว่าเทอร์มินัล CRT ถูกต่อกับไมโครคอมพิวเตอร์ผ่านทางการอินเตอร์เฟสแบบ RS-232-C เรามีวิธีใดในการตัดสินว่าอุปกรณ์ตัวใดเป็น DCE หรือ DTE คำตอบก็คือจากที่ได้ทราบมาแล้วว่า RS-232-C ใช้ในการสื่อสารข้อมูลระหว่าง DCE และ DTE ดังนั้นอุปกรณ์ตัวใดตัวหนึ่งต้องเป็น DTE หรือ DCE เพื่อให้เข้าคู่กันได้ ตามปรกติ CRT และเทอร์มินัลที่เป็นตัวส่งหรือตัวรับข้อมูลจะถูกกำหนดเป็น DTE แต่ไมโครคอมพิวเตอร์สามารถเป็นได้ทั้ง DCE และ DTE ตัวอย่างเช่น พอร์ท I/O ของเครื่อง Cromemco เป็น DCE แต่พอร์ท I/O แบบอนุกรมของ TRS-80 เป็น DTE แสดงว่าคอมพิวเตอร์ของแต่ละบริษัทนั้นกำหนดลักษณะของพอร์ทแตกต่างจากกันออกไป



รูปที่ 2.12 ลักษณะของระบบที่ใช้ในตัวอย่างที่ 2

2.5.2 คุณสมบัติของสัญญาณไฟฟ้า

เราจะอธิบายคุณสมบัติของสัญญาณไฟฟ้าของ RS-232-C ในหัวข้อนี้โดยอาศัยรูปที่ 2.13 ประกอบดังนี้



รูปที่ 2.13 คุณสมบัติทางไฟฟ้าของการอินเตอร์เฟสแบบ RS-232-C

ซึ่งจะสังเกตพบว่า

- สัญญาณที่ขาทุกขาของคอนเน็คเตอร์ของ RS-232-C จะเป็นสภาวะใดสภาวะหนึ่งในแต่ละคู่ต่อไปนี้
  - MARK / SPACE
  - ON / OFF
  - LOGIC 0 / LOGIC 1

ความสัมพันธ์ระหว่างสถานะของสัญญาณคู่ต่างๆ กับระดับแรงดันได้แสดงไว้ในตารางที่ 2.4 ขอให้สังเกตด้วยว่า RS-232-C ใช้ระดับลอจิกลบแทนระดับแรงดันต่างๆ (ลอจิกลบ (negative logic) คือวิธีการเปรียบเทียบระดับแรงดันแบบหนึ่ง ถ้าระดับแรงดันหนึ่งค่าเป็นลบมากกว่าอีกระดับแรงดันหนึ่ง ระดับแรงดันที่มีค่าเป็นลบมากกว่าจะเป็นลอจิกสูง กล่าวคือ 1= -V หรือ OFF ส่วน 0= +V หรือ ON) โดยแรงดันของระดับสัญญาณต่างๆ จะถูกวัดเทียบกับเซอร์กิต signal ground นอกจากนี้ช่วงของระดับแรงดันระหว่าง -3 ถึง +3 โวลต์ จะเป็นช่วงของการเปลี่ยนแปลงลอจิก (Transition) ดังนั้นจึงไม่มีการระบุสถานะของสัญญาณในช่วงนี้

ตารางที่ 2.4 แสดงความสัมพันธ์ระหว่างสถานะของสัญญาณคู่ต่าง ๆ กับระดับแรงดัน

Status	**Signal Voltage**	
	-25V < V1 < -3V	**3V < V1 < 25V**
Binary Logic	1	0
Signal Condition	MARK	SPACE
Function	OFF	ON

- ในการแทนลอจิกหนึ่งหรือสถานะ MARK ตัวขับสัญญาณ (driver) ต้องจ่ายแรงดันระหว่าง -5 ถึง -15 โวลต์ ส่วนในการแทนลอจิก 0 หรือ SPACE ตัวขับสัญญาณต้องจ่ายแรงดันระหว่าง +5 ถึง +15 โวลต์

จากทั้งสองกรณีข้างต้นแสดงว่า RS-232-C ยอมให้มี noise margin ได้ไม่เกิน 2 โวลต์ สำหรับความสัมพันธ์ระหว่างระดับแรงดันและสถานะของสัญญาณได้แสดงไว้ในรูปที่ 2.13 จากรูปจะเห็นได้ว่าถ้า line driver หรือตัวกำเนิดสัญญาณต้องการส่งลอจิก 0 line driver จะต้องจ่ายแรงดันระหว่าง +5 ถึง +15 โวลต์ ส่วน line receiver หรือตัวรับสัญญาณปลายทางจะถือว่าแรงดันที่อยู่ภายในช่วง +3 ถึง +15 โวลต์แทนลอจิก 0 จากการเปรียบเทียบระดับสัญญาณของตัวส่งและตัวรับจะเห็นว่า RS-232-C ยอมให้มีการ drop ของสัญญาณในช่วง 2 โวลต์เกิดขึ้นได้ สำหรับในด้านการส่งลอจิก 1 ก็เป็นเช่นเดียวกัน

จากที่ได้อธิบายมาอาจมีข้อสงสัยว่าทำไมไม่ใช้สถานะลอจิกแบบ TTL ซึ่งระดับของแรงดันมีค่าระหว่าง 0 ถึง +5 โวลต์ และทำไมถึงต้องใช้ระดับแรงดันระหว่าง -15 ถึง -3 และ +3 ถึง +15 โวลต์ด้วย

สาเหตุที่ไม่ใช้การแทนลอจิกแบบ TTL ก็เพราะสถานะลอจิกแบบ TTL ถูกรบกวนจากสัญญาณรบกวนต่างๆ ได้ง่าย นอกจากนี้ยังมีปัญหาเกี่ยวกับระยะทางที่สามารถทำการสื่อสารข้อมูลอีกด้วย สำหรับสาเหตุที่ต้องใช้แรงดันในช่วง -15 ถึง-3 และ +3 ถึง +15 ก็เพราะในขณะที่กำลังทำการพัฒนา RS-232-C นั้นในวงจรคอมพิวเตอร์ต่างๆ โดยทั่วไปมีการใช้ระดับแรงดันในช่วงเหล่านี้อยู่ อนึ่งทรานซิสเตอร์ที่มีขายทั่วๆ ไปสามารถทำงานได้ในช่วงแรงดันเหล่านี้และยังทนต่อสัญญาณรบกวนต่างๆ ที่มีเข้ามาได้ นอกจากนี้ยังสามารถทำงานที่ความถี่สูงๆ ได้โดยสูงถึง 20,000 บิทต่อวินาที (bps) ยิ่งกว่านั้นสถานะ MARK และสถานะ SPACE ยังถูกแทนด้วยการไหลของกระแสในทิศทางที่ตรงข้ามกันและความแตกต่างของแรงดันที่สถานะ MARK และ SPACE มีค่าสูงถึง 6 โวลต์เป็นอย่างน้อย ข้อดีต่างๆ ที่กล่าวมานี้ช่วยให้การส่งข้อมูลมีเสถียรภาพดี

- ตัวเก็บประจุ ที่ต่อขนานกับอุปกรณ์รับข้อมูลปลายทางจะต้องมีค่าไม่เกิน 2,500 pF โดยค่านี้ไม่รวมค่าความจุไฟฟ้าของสายเคเบิ้ลเข้าไปด้วย

หมายเหตุ: ตามข้อกำหนดนี้ระยะทางที่สามารถใช้ทำการสื่อสารข้อมูลได้ต้องไม่เกิน 50 ฟุต ซึ่งถูกกำหนดไว้ในมาตรฐาน RS-232-C

- แรงดันขณะเปิดวงจรหรือขณะที่ไม่มีโหลดจะต้องไม่เกิน 25 โวลต์
- วงจรรับสัญญาณที่ใช้กับ RS-232-C ต้องสามารถทนต่อการลัดวงจรที่เกิดขึ้นได้ เช่น ขา 2 ขาเกิดการลัดวงจรโดยไม่ได้ตั้งใจ โดยไม่ทำให้เกิดความเสียหายต่อตัวมันเองหรืออุปกรณ์ที่เกี่ยวข้องด้วย เช่น เทอร์มินัล , โมเด็ม พอร์ท I/O และอุปกรณ์ต่างๆ ที่ต่อเข้ากับเคเบิ้ลที่ใช้ในการอินเตอร์เฟสแบบ RS-232-C

2.5.3 คุณสมบัติทางกลของการอินเตอร์เฟส

รายละเอียดของขาต่างๆ ของคอนเนคเตอร์ตามมาตรฐาน RS-232-C ได้แสดงไว้ในตารางที่ 1ผ ในภาคผนวก ขอให้สังเกตด้วยว่าตามมาตรฐาน RS-232-C ไม่ได้กล่าวถึงปลั๊กตัวผู้ (plug) หรือปลั๊กตัวเมีย (socket) ของคอนเนคเตอร์เลยว่าต้องมีรูปร่างลักษณะอย่างไร ในปัจจุบันเรามักจะใช้คอนเนคเตอร์แบบ DB-25 (บางทีเรียกแบบ D-type 25 pin connector) ในการอินเตอร์เฟสตามมาตรฐาน RS-232-C คอนเนคเตอร์แบบนี้เทียบเท่ากับแบบ ISO 1123 ซึ่งเป็นมาตรฐานที่ประกาศใช้โดย International Organization for Standardization (ISO)

2.5.4 ลักษณะการทำงานของเซอร์กิตต่างๆ

เซอร์กิตต่างๆ ได้แสดงไว้ในตารางที่2ผ ในภาคผนวกซึ่งสามารถแยกออกเป็นประเภทต่างๆ ได้ 5 ประเภทคือ

- กราวนด์ หรือ Common Return [A]
- เซอร์กิตข้อมูล [B]
- เซอร์กิตควบคุม [C]
- เซอร์กิตของสัญญาณฐานเวลา (timing circuit) [D]
- เซอร์กิตของแชนแนลที่ 2 [S]

จากตารางที่ 2ผ จะเห็นได้ว่าตัวอักษรในวงเล็บที่อยู่ท้ายเซอร์กิตประเภทต่างๆ จะเป็นตัวอักษรตัวแรกของกลุ่มตัวอักษร (ประกอบด้วยตัวอักษรสองหรือสามตัว) ซึ่งใช้กันทั่วๆ ไปในการอธิบายสัญญาณต่างๆ ที่เกี่ยวข้องกับการใช้งาน RS-232-C จากตารางที่ 2ผ เซอร์กิตต่างๆ ของ RS-232-C ถูกแบ่งออกเป็นประเภทๆ โดยใช้กลุ่มของตัวอักษรดังอธิบายไว้ข้างต้นนอกจากนี้ในตารางยังแสดงทิศการเคลื่อนที่ของข้อมูล สัญญาณควบคุม และสัญญาณเวลาว่าส่งจาก DCE หรือส่งจาก DTE รวมทั้งกำหนดขาสัญญาณที่ใช้กำกับเซอร์กิตต่างๆ สำหรับลักษณะการทำงานของเซอร์กิตต่างๆ มีดังนี้

2.5.4.1 Circuit AA : Protective Ground

ลวดตัวนำของเซอร์กิตนี้จะถูกต่อเข้ากับตัวถังของอุปกรณ์เพื่อใช้เป็นสายดิน เมื่อเปรียบเทียบ protective ground กับ signal ground จะเห็นได้ว่า signal ground มีความสำคัญกว่ามากดังนั้น protective ground จึงมักไม่ถูกต่อ การกระทำเช่นนี้ไม่เป็นการทำผิดข้อกำหนดมาตรฐาน RS-232-C เนื่องจากว่า RS-232-C ได้กำหนดให้กรณีนี้เป็นกรณีเลือกใช้งาน (option)

2.5.4.2 Circuit AB : Signal หรือ Common Return

เซอร์กิตนี้ถูกใช้เป็นจุดอ้างอิงของสัญญาณที่ใช้ในเซอร์กิตต่างๆ ยกเว้นเซอร์กิต AA (protective ground) เซอร์กิตนี้เป็นเซอร์กิตเดียวที่ต้องถูกต่อไว้เสมอไม่ว่าเป็นการประยุกต์ใช้งานแบบใด

2.5.4.3 Circuit BB : Received Data

สัญญาณของเซอร์กิตนี้จะถูกส่งจาก DTE ไปยัง DCE โดย DTE จะทำให้เซอร์กิต BA (transmitted Data) มีสถานะลอจิกเป็น 1 (MARK) ตลอดเวลาที่ไม่มีการส่งข้อมูล

ในระบบทุกระบบที่ใช้มาตรฐาน RS-232-C DTE จะไม่ทำการส่งข้อมูลนอกจากว่าเซอร์กิตต่อไปนี้มีสถานะลอจิกเป็น 0

- เซอร์กิต CA (Request to send)
- เซอร์กิต CB (Clear to send)
- เซอร์กิต CC (Data set ready)
- เซอร์กิต CD (Data terminal ready)

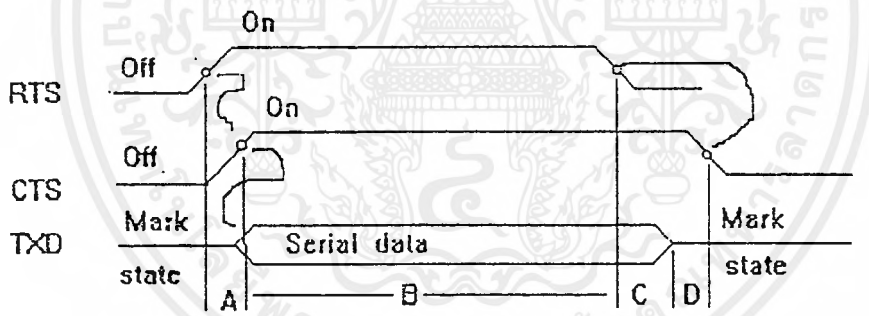
สำหรับการทำงานร่วมกันของเซอร์กิตเหล่านี้จะอธิบายในหัวข้อถัดไป

ในบางกรณีเราอาจต้องใช้ null modem ร่วมกับเซอร์กิตนี้ ตัวอย่างเช่น ในระบบคอมพิวเตอร์ระบบหนึ่งเราทำการอินเตอร์เฟสคอมพิวเตอร์ของเราเข้ากับเทอร์มินัลผ่านทาง RS-232-C ถ้าเทอร์มินัลเป็น DTE และคอมพิวเตอร์ทำตัวเป็น DCE คอมพิวเตอร์จะคอยรับสัญญาณที่ส่งจากเทอร์มินัลผ่านทางสาย Transmitted Data (การส่งนี้เราใช้ DTE เป็นตัวอ้างอิงไม่ใช่ DCE) แต่ถ้าคอมพิวเตอร์ทำตัวเป็น DTE และส่งข้อมูลผ่านทางเซอร์กิตนี้ก็จะเกิดปัญหาขึ้นคือ อุปกรณ์ทั้งสองตัวจะส่งข้อมูลลงบนสายเส้นเดียวกันทำให้ข้อมูลเกิดการต้านกัน จึงต้องใช้ null modem ช่วยแก้ปัญหานี้

ปัญหาที่สำคัญอีกข้อหนึ่งก็คือในเซอร์กิตบางเซอร์กิตซึ่งจะต้องอยู่ในสภาวะ ON ก่อนที่จะมีการส่งข้อมูลแต่กลับไม่อยู่ในสภาวะ ON สาเหตุนี้อาจเกิดจากการทำงานที่ผิดพลาดของตัวเทอร์มินัล , พอร์ท I/O ของคอมพิวเตอร์ หรือเกิดจากตัวสายเคเบิ้ลที่ต่อระหว่างเทอร์มินัลกับคอมพิวเตอร์ ถ้ามีปัญหาเหล่านี้เกิดขึ้นในขณะที่เราส่งข้อมูลโดยใช้ RS-232-C ปัญหาของเราอาจเกิดจากสัญญาณใดสัญญาณหนึ่งข้างต้นเป็น OFF หรือเทอร์มินัลของเราทำการส่งข้อมูลบนสายเส้นเดียวกับที่คอมพิวเตอร์ใช้ส่งข้อมูล

2.5.4.4 Circuit CA : Request to send

สัญญาณ Request to send (RTS) และสัญญาณ Clear to send (CTS) ซึ่งเกิดขึ้นในการส่งข้อมูลระหว่าง DTE และ DCE แสดงไว้ในรูปที่ 2.14 และรูปที่ 2.15



รูปที่ 2.14 การทำแฮนด์เช็คของสัญญาณ RTS และสัญญาณ CTS

ในการส่งข้อมูลแบบ simplex และ full-duplex เมื่อ request to send มีสถานะของลอจิกเป็น 0 (ON) จะทำให้ DCE อยู่ในโหมดการส่งข้อมูล (transmit mode) แต่ถ้าเป็นกรณีการส่งข้อมูลแบบ half-duplex เมื่อ request to send อยู่ในสถานะ ON DCE จะอยู่ในโหมดการส่งข้อมูล แต่ถ้า request to send อยู่ในสภาวะ OFF DCE จะอยู่ในโหมดการรับข้อมูล (คือ DCE จะรับข้อมูลจากเครือข่ายการสื่อสารและส่งข้อมูลเหล่านี้ไปยัง DTE)

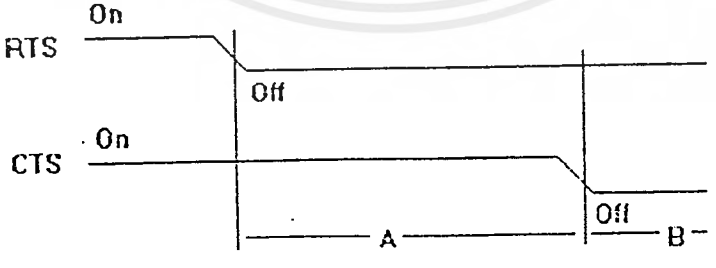
สำหรับในรูปที่ 2.14 จะสามารถอธิบายได้ว่าการทำ handshake ของสัญญาณ RTS และ CTS ในช่วง A DTE จะป้อนสัญญาณ RTS แสดงให้ DCE ทราบว่า DTE ต้องการส่งข้อมูลซึ่งจะเกิดขั้นตอนดังนี้คือ DCE จะจัดตั้งช่องทางการสื่อสารและป้อนสัญญาณ CTS (เป็น ON) ซึ่งแสดงให้ DTE ทราบว่า DTE สามารถเริ่มต้นส่ง

ข้อมูลได้แล้ว แต่ TXD จะยังอยู่ในสภาวะ MARK อยู่ ในช่วง B ข้อมูลจะถูกส่งผ่านทางเซอร์กิต transmitted data เมื่อข้อมูลถูกส่งออกไปจนหมดแล้ว DTE จะ OFF สัญญาณ RTS เพื่อบอกให้ DCE ทราบว่า DTE ไม่ต้องการส่งข้อมูลอีก ต่อไปในช่วง C เมื่อ DCE ส่งข้อมูลทั้งหมดออกไปยัง communication link เรียบร้อยแล้ว เซอร์กิต TXD จะกลับเข้าสู่สภาวะ MARK ในช่วง D DCE จะแจ้งให้ DTE ทราบว่า DCE พร้อมแล้วที่จะรับข้อมูลชุดใหม่เพื่อส่งออกไปโดยการ OFF สัญญาณ CTS

จากรูปที่ 2.14 ขอให้ดูที่สภาวะการเปลี่ยนแปลงจาก ON ไปเป็น OFF และ OFF ไปเป็น ON ของสัญญาณ request to send การเปลี่ยนสภาวะของสัญญาณ request to send จาก OFF เป็น ON จะทริก DCE ให้อยู่ในโหมดการส่งข้อมูล และทำขั้นตอนต่างๆ ที่ให้การสื่อสารข้อมูลเกิดขึ้นได้ เช่น การต่อโทรศัพท์ไปยังคอมพิวเตอร์หลัก (host computer) (ถ้า DCE ตัวนั้นสามารถต่อโทรศัพท์โดยอัตโนมัติได้) ในระบบไมโครคอมพิวเตอร์ทั่วๆ ไปสาย request to send จะถูกต่ออยู่กับสาย clear to send โดยตรงดังนั้นเมื่อไรก็ตามที่ DTE ป้อนสัญญาณ request to send DTE ก็จะได้รับสัญญาณตัวนี้กลับมาโดยส่งมาทางสาย clear to send เมื่อขั้นตอนต่างๆ ที่ทำให้เกิดการสื่อสารข้อมูลได้เกิดขึ้นเรียบร้อยแล้ว DCE จะทำให้ clear to send (เซอร์กิต CB) มีค่าลอจิกเป็น 0 (ON) ซึ่งเป็นการบอกให้ DTE ทราบว่าสามารถส่งข้อมูลทางเซอร์กิต transmit data ข้ามจุดเชื่อมต่อ (interfacing point) ได้แล้ว

การเปลี่ยนสถานะจาก ON ไปเป็น OFF ของสาย RTS เป็นการส่งให้ DCE ส่งข้อมูลที่ยังเหลืออยู่ที่จุดเชื่อมต่อของเซอร์กิต transmit dataออกไปยังช่องทางการสื่อสารและออกจากโหมดการส่งข้อมูล(ในกรณีของ full-duplex หรือ simplex) หรือเข้าสู่โหมดการรับข้อมูล(ในกรณีของ half-duplex) DCE จะตอบสนองต่อสัญญาณนี้โดยทำให้สัญญาณ clear to send มีลอจิกเป็น 1 (OFF)

จากรูปที่ 2.14 กล่าวได้ว่าเมื่อใดก็ตามที่ request to send เปลี่ยนสถานะจาก ON เป็น OFF request to send จะ ON ใหม่ได้อีกครั้งก็ต่อเมื่อ DCE สั่งให้ clear to send เปลี่ยนสถานะจาก ON เป็น OFF แล้ว การทำเช่นนี้เป็นการป้องกันไม่ให้เกิด overrun error ขึ้น ซึ่งก็คือ DTE ทำการส่งข้อมูลชุดใหม่มาอีก ในขณะที่ DCE ยังส่งข้อมูลชุดเก่าไม่เรียบร้อย



รูปที่ 2.15 การ ON สัญญาณ RTS ใหม่ของ DTE

จากรูปที่ 2.15 จะเห็นได้ว่าในช่วง A DTE ไม่สามารถ ON สัญญาณ RTS ใหม่อีกครั้งหนึ่งได้ DTE ต้องรอจนกระทั่ง DCE ส่งข้อมูลที่เหลือออกไปจนหมดโดย DCE จะ OFF สัญญาณ CTS เพื่อแสดงให้ DTE ทราบว่ามันพร้อมที่จะรับข้อมูลชุดใหม่แล้วในช่วง B DTE สามารถ ON สัญญาณ RTS ใหม่เมื่อใดก็ได้เนื่องจาก CTS มีสภาวะเป็น OFF

การทำแฮนด์เชคโดยใช้ request to send กับ clear to send ที่อธิบายไปนั้นใช้ได้ทั้งในการส่งข้อมูลทีละอักขระหรือทีละบล็อค ตัวอย่างเช่น สมมติให้ 1 อักขระประกอบด้วยบิทต่างๆ 10 บิท DTE จะป้อนสัญญาณลงในสาย request to send และคอยรับสัญญาณจาก DCE ทางสาย clear to send สำหรับการทำแฮนด์เชคในการส่งข้อมูลทีละบล็อคนั้น DTE จะส่งอักขระพิเศษที่บอกจุดสิ้นสุดของบล็อค (end of transmission character) เพิ่มเข้าไปด้วย และเมื่อถึงจุดสิ้นสุดของบล็อคข้อมูล DTE จะ OFF สัญญาณ request to send ในการตอบสนองต่อเหตุการณ์เหล่านี้ DCE จะ OFF สัญญาณ clear to send เมื่ออักขระพิเศษที่บอกจุดสิ้นสุดของข้อมูลได้ถูกส่งจาก DCE ออกไปยังเครือข่ายการสื่อสารเรียบร้อยแล้ว

จากที่ได้อธิบายไว้ในเซอร์กิต BA , DTE จะส่งข้อมูลได้ก็ต่อเมื่อ request to send , clear to send , data set ready และ data terminal ready ON โดยเมื่อสัญญาณ data set ready ON เรียบร้อยแล้วสัญญาณ request to send จะ ON ตามมาและ DCE จะตอบสนองต่อสัญญาณนี้โดยการส่งสัญญาณ clear to send เป็น ON กลับมายัง DTE DTE จะส่งข้อมูลออกทางเซอร์กิต transmit data ได้ (เราสามารถ ON สัญญาณ request to send เมื่อไรก็ตามที่ clear to send เป็น OFF โดยไม่ต้องสนใจสภาวะของเซอร์กิตอื่นๆ)

สิ่งที่น่าสนใจอีกข้อหนึ่งคือสัญญาณ request to send เป็นสัญญาณที่ใช้ระหว่าง DTE และ DCE เท่านั้น เนื่องจากว่าการเชื่อมโยงโดยสายโทรศัพท์ , ไมโครเวฟ หรือดาวเทียมจะแยก DCE ทางด้านผู้ส่ง (local) และผู้รับ (remote) ออกจากกัน ดังนั้นสัญญาณนี้จะไม่ถูกส่งไปยังอุปกรณ์ใดๆ ที่อยู่ทางด้านที่เราจะติดต่อด้วย (remote) และสัญญาณนี้ก็ไม่ได้บอกสภาวะของเครื่องทางด้านผู้รับด้วยเช่นกัน

การใช้งานเซอร์กิต request to send ยังเป็นตัวก่อให้เกิดปัญหาในการอินเตอร์เฟสขึ้นอีกด้วย โดยเฉพาะในการใช้งานร่วมกับระบบไมโครคอมพิวเตอร์ที่ใช้อุปกรณ์พวก USART (Universal Synchronous/Asynchronous Receiver/Transmitter) เป็นพอร์ท I/O แบบอนุกรม

2.5.4.5 Circuit CB : clear to send

สัญญาณเป็นสัญญาณควบคุมซึ่งถูกส่งจาก DCE ไปยัง DTE เพื่อบอกให้ DTE ทราบว่า DCE พร้อมที่จะรับข้อมูลที่จะส่งมาจาก DTE บนสาย transmit data แล้วเมื่อสัญญาณ clear to send อยู่ในสภาวะ ON รวมทั้งสัญญาณ request to send , data set ready หรือ data teminal ready มีสภาวะเป็น ON ด้วย การ ON ของสัญญาณเหล่านี้จะบอกให้ DTE ทราบว่าข้อมูลที่ส่งไปยัง DCE จะถูก DCE รับไว้และส่งต่อไปยังช่องสัญญาณการสื่อสาร เมื่อสัญญาณ clear to send อยู่ในสภาวะ OFF จะบอกให้ DTE ทราบว่า DCE ไม่พร้อมที่จะรับข้อมูลดังนั้น DTE จะยังไม่ส่งข้อมูลออกมา (ต้องป้อนสัญญาณ RTS ใหม่)

clear to send จะอยู่ในสภาวะ ON ก็ต่อเมื่อสัญญาณ request to send (เซอร์กิต CA) และ data set ready (เซอร์กิต CC) จะอยู่ในสภาวะ ON ทั้งคู่ ถ้าไม่ใช้ขา request to send ให้ถือว่าสัญญาณ request to send เป็น

ON ตลอดเวลา ดังนั้นสภาพสัญญาณของ clear to send จะเป็นอย่างไรจึงขึ้นอยู่กับสถานะของสัญญาณ data set ready ว่าเป็น ON หรือ OFF

ในการอินเตอร์เฟสตามมาตรฐานของ RS-232-C ซึ่งเป็นการทำอินเตอร์เฟสระหว่าง DTE และ DCE ในกรณีที่มีเครือข่ายสวิชชิ่งโทรศัพท์เข้าไปเกี่ยวพันด้วย เราจะต้องใช้เซอร์กิตต่อไปนี้เพิ่มเติมเข้าไปด้วย

- เซอร์กิต CC : data set ready
- เซอร์กิต CD : data terminal ready
- เซอร์กิต CE : ring indicator
- เซอร์กิต CF : received line signal detector

เนื่องจากการทำงานของเซอร์กิตต่อไปนี้เกี่ยวพันกับเครือข่ายโทรศัพท์ ดังนั้นถ้าเราทำการติดต่อในระยะสั้นๆ (ไม่ผ่านเครือข่ายโทรศัพท์) โดยใช้ RS-232-C เพียงอย่างเดียวสามารถตัดเซอร์กิตเหล่านี้ออกไปได้ การประยุกต์ใช้งานที่น่าสนใจคือการใช้งานที่ร่วมกับอุปกรณ์ที่สามารถต่อโทรศัพท์ได้โดยอัตโนมัติ (auto dial) และอุปกรณ์ที่สามารถตอบรับต่อการเรียกโดยอัตโนมัติ

สำหรับหน้าที่สำคัญของสัญญาณ data set ready และ data terminal ready คือมันเป็นตัวแสดงให้เราทราบว่าอุปกรณ์ของเราพร้อมที่จะทำการสื่อสารข้อมูลหรือไม่ โดยถ้า data set ready อยู่ในสภาวะ ON จะบอกให้ DTE ทราบว่า DCE พร้อมที่จะส่งข้อมูลที่รับจาก DTE ออกไปยังเครือข่ายการสื่อสารแล้ว (เช่น เครือข่ายโทรศัพท์ , ไมโครเวฟ) ในลักษณะเดียวกันถ้า data terminal ready อยู่ในสภาวะ ON แสดงว่า DTE พร้อมจะส่งข้อมูลไปให้ DCE โดยส่งข้อมูลออกทางเซอร์กิต transmit data ดังที่ได้อธิบายไว้แล้วในข้างต้น ทั้ง data set ready และ data terminal ready จะต้องอยู่ในสภาวะ ON ก่อนที่จะมีการส่งข้อมูลเกิดขึ้นแต่เนื่องจากสัญญาณเหล่านี้ไม่ค่อย ถูกใช้ในกรณีที่ทำการอินเตอร์เฟสกับระบบคอมพิวเตอร์ ดังนั้นข้อมูลจะถูกส่งเมื่อใดจึงขึ้นอยู่กับสภาวะของสัญญาณ request to send และ clear to send

2.5.4.6 Circuit CC : data set ready

สัญญาณนี้เป็นสัญญานควบคุมที่ส่งจาก DCE ไปยัง DTE ในกรณีที่ data set ready อยู่ในสภาวะ ON แสดงว่า DCE ได้ถูกต่อกับช่องสัญญาณการสื่อสารเรียบร้อยแล้ว ในกรณีที่ DCE ของเราสามารถต่อกับโทรศัพท์ได้โดยอัตโนมัตินั้นการที่ data set ready เป็น ON หมายความว่า DCE ของเรา (local) ได้ต่อโทรศัพท์(เรียก) DCE ของเครื่องคอมพิวเตอร์ที่เราต้องการติดต่อด้วย (remote) ได้ตอบรับต่อการเรียกทำให้เกิดการเชื่อมต่อกันของช่องสัญญาณการสื่อสารขึ้นระหว่าง DCE ทั้งสองด้านทำให้สามารถทำการสื่อสารข้อมูลระหว่างกันได้ เมื่อช่องสัญญาณการสื่อสารถูกเชื่อมต่อแล้วระบบจะเข้าสู่โหมดการส่งข้อมูล (ไม่ใช่โหมดการส่งสัญญาณเสียง คือเราพูดสายไม่ได้)

2.5.4.7 Circuit CD : data terminal ready

สัญญาณควบคุมตัวนี้จะถูกส่งจาก DTE ไปยัง DCE สัญญาณ data terminal ready ต้องอยู่ในสภาวะ ON ก่อนที่ DCE จะ ON สัญญาณ data set ready (ซึ่งบอกให้รู้ว่า DCE ถูกเชื่อมต่อเข้ากันช่องสัญญาณการสื่อ

สารแล้ว และสามารถส่งข้อมูลได้) ขณะใดก็ตามที่ DCE ต่อกับช่องสัญญาณการสื่อสารแล้ว data terminal ready เปลี่ยนสภาวะจาก ON เป็น OFF DCE จะตัดการเชื่อมต่อระหว่าง DCE กับช่องสัญญาณการสื่อสารทิ้งในทันที

2.5.4.8 Circuit CE : ring indicator

สัญญาณนี้เป็นสัญญาณควบคุมที่ส่งจาก DCE ไปยัง DTE เมื่อสัญญาณนี้มีสภาวะเป็น ON แสดงว่า DCE กำลังได้รับสัญญาณเสียงกริ่ง และในกรณีที่ DCE ไม่ได้รับสัญญาณเสียงกริ่งสัญญาณ ring indicator จะมีสภาวะเป็น OFF เราใช้สัญญาณควบคุมนี้ในกรณีที่ใช้โมเด็มที่สามารถตอบรับการต่อเรียกได้โดยอัตโนมัติ (autoanswer)

2.5.4.9 Circuit CF : received line signal detector

สัญญาณนี้ส่งจาก DCE ไปยัง DTE เมื่อ DCE ได้รับสัญญาณ carrier ที่ส่งมาจาก DCE อีกด้านหนึ่ง สัญญาณ received line signal detector จะมีสภาวะเป็น ON แสดงว่า DCE จับสัญญาณในช่องสัญญาณการสื่อสารซึ่งจะถูกนำไปดีมอทดูเลทได้แล้ว ในโมเด็มแบบต่างๆ สายเส้นนี้จะถูกต่อกับ LED เพื่อแสดงให้รู้ว่าขณะนี้มีสัญญาณ carrier เข้ามาหรือไม่

2.5.5 โครงสร้างทั่วๆ ไปของมาตรฐาน RS-232-C

ดังที่ได้กล่าวมาแล้วในตอนต้น มาตรฐาน RS-232-C ไม่ได้กำหนดลักษณะโครงสร้างของคอนเนคเตอร์ไว้เลย ดังนั้นเราสามารถเลือกใช้คอนเนคเตอร์แบบใดก็ได้ อย่างไรก็ตาม EIA ระบุไว้ว่าคอนเนคเตอร์ที่เข้ามาตรฐานของทหารแบบ MIL-C-24308(MS-18275) สามารถใช้ในการอินเตอร์เฟสตามมาตรฐาน RS-232-C ได้เป็นอย่างดีแต่ก็ไม่ได้เป็นส่วนหนึ่งของมาตรฐาน RS-232-C แต่อย่างใด

จากมาตรฐาน RS-232-C เราจะเห็นว่ามีการใช้สัญญาณต่างๆ อยู่เป็นจำนวนมากดังนั้นในการใช้งานเราอาจสงสัยว่าควรจะใช้สัญญาณเส้นใดบ้างจึงจะเหมาะสมกับระบบของเรา(จากทั้งหมด 21 เซอร์กิต) ข้อสงสัยนี้ตอบได้ยากมากเนื่องจากรูปแบบของกลุ่มสัญญาณที่ใช้ในการสื่อสารข้อมูลมีแตกต่างกันออกไปมากมายขึ้นอยู่กับลักษณะของระบบที่ต้องการใช้ โครงสร้างของระบบอาจเป็นไปได้หลายแบบตั้งแต่การต่อเทอร์มินับเข้ากับเครื่องคอมพิวเตอร์ซึ่งเป็นการอินเตอร์เฟสแบบง่ายๆ , ใช้เครื่องคอมพิวเตอร์ของเราร่วมกับเครื่องคอมพิวเตอร์เครื่องอื่นโดยอาศัยการมัลติเพล็กซ์ , ใช้ส่งและรับข้อมูลแบบซิงโครนัส หรือใช้กับสาย dedicated line(สายที่ใช้ในการส่งข้อมูลเพียงอย่างเดียว ใช้ส่งสัญญาณเสียงไม่ได้) ซึ่งเป็นการใช้ร่วมกับเครื่องเทอร์มินัลปลายทางเครื่องอื่นๆ อย่างไรก็ตาม EIA ได้แบ่งมาตรฐานของการใช้สายสัญญาณออกเป็นกลุ่มตามสภาพของระบบต่างๆ กลุ่มของสัญญาณที่ใช้ร่วมกับระบบไมโครคอมพิวเตอร์ถูกแบ่งออกเป็น 7 กลุ่มดังนี้

- ใช้ในการส่งข้อมูลอย่างเดียว
- ใช้ในการส่งข้อมูลอย่างเดียวแต่ใช้สัญญาณ RTS ด้วย
- ใช้ในการรับข้อมูลอย่างเดียว
- ใช้ส่งและรับข้อมูลแบบ half-duplex
- ใช้ส่งและรับข้อมูลแบบ full-duplex

- ใช้ส่งและรับข้อมูบแบบ full-duplex แต่ใช้สัญญาณ RTS ด้วย

- แบบพิเศษ

การอินเตอร์เฟสแบบต่างๆ ที่กล่าวมานี้ได้แสดงไว้ในตารางที่ 3ผ ในภาคผนวกซึ่งได้ระบุสายสัญญาณที่ต้องต่อในการอินเตอร์เฟสแต่ละระบบ (เราได้ตัดสัญญาณที่ใช้ในการส่งข้อมูลแบบซิงโครนัสทิ้งไป เนื่องจากว่าในระบบไมโครคอมพิวเตอร์มักจะไม่ใช้การส่งข้อมูลแบบนี้)โดยที่

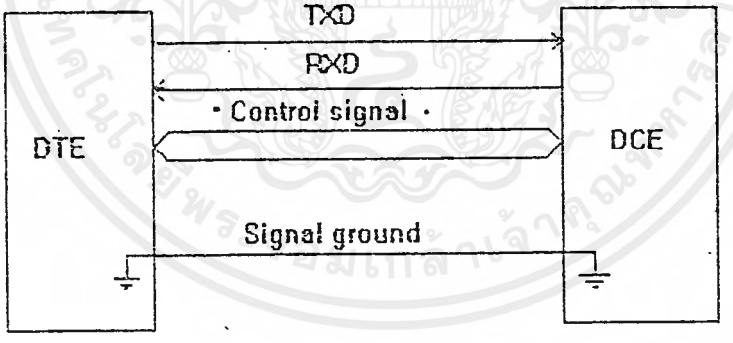
X = ใช้งานในทุกกรณี

S = ใช้ในกรณีที่ระบบของเราเกี่ยวพันกับเครือข่ายโทรศัพท์ด้วย

O= เลือกใช้ตามลักษณะของระบบเรา

จากตารางที่ 3ผ จะเห็นว่ามีเซอร์กิตอยู่เซอร์กิตหนึ่งที่ต้องต่ออยู่เสมอคือ เซอร์กิต signal ground (ขา7) ส่วนเซอร์กิตอื่นจะต่อหรือไม่ขึ้นกับการประยุกต์ใช้งานของเราโดยให้เลือกจากวิธีการส่งข้อมูลของเราว่าเป็นแบบใดและต้องดูด้วยว่าเข้ากับระบบของเราได้หรือไม่

ในการส่งข้อมูลแบบ full-duplex นั้น ระบบไมโครคอมพิวเตอร์บางระบบอาจใช้เซอร์กิตต่างออกไปจากที่ได้แสดงไว้ในตารางที่ 3ผ ในระบบไมโครคอมพิวเตอร์ส่วนใหญ่มักใช้อุปกรณ์ที่ส่งข้อมูลในทิศทางเดียว เช่น ใช้รับข้อมูลอย่างเดียวก็คือเครื่องพิมพ์ หรือส่งข้อมูลอย่างเดียว เช่น จอยสติ๊ก แต่เนื่องจากระบบเคเบิ้ลแบบ full-duplex ตามมาตรฐาน RS-232-C ถูกออกแบบให้ใช้ส่งข้อมูลได้ใน 2 ทิศทาง ดังนั้นเราสามารถใช้สายเคเบิ้ลกับอุปกรณ์ที่ใช้รับข้อมูลอย่างเดียว เช่น เครื่องพิมพ์ หรือจะนำไปใช้กับอุปกรณ์ตัวอื่นได้ในทันทีโดยไม่ต้องแก้ไขเพิ่มเติมอะไรอีก



รูปที่ 2.16 ลักษณะพื้นฐานของการส่งข้อมูลแบบ full-duplex

การประยุกต์ใช้งานของการส่งข้อมูลแบบ full-duplex ที่ใช้กันมากในระบบไมโครคอมพิวเตอร์คือ ใช้กับเทอร์มินัลที่ใช้ทำหน้าที่รับและส่งข้อมูลกล่าวคือ อักขระจะถูกส่งจากคีย์บอร์ดไปยังไมโครคอมพิวเตอร์และถูกสะท้อนไปแสดงบนจอภาพหรือพิมพ์ออกที่เครื่องพิมพ์ ในกรณีเช่นนี้ ข้อมูลจะเคลื่อนที่ใน 2 ทิศทางพร้อมๆ กันโดยจะถูกส่งและรับจาก DTE(คีย์บอร์ดและจอภาพ) กับ DCE (พอร์ทอินพุท/เอาท์พุท แบบอนุกรมของไมโครคอมพิวเตอร์) ดังแสดงในรูปที่ 2.16

2.6 การใช้งานของพอร์ตอนุกรมของ MCS-51

2.6.1 การสื่อสารข้อมูลอนุกรม

การสื่อสารข้อมูลอนุกรมเป็นการรับหรือการส่งข้อมูลในลักษณะกลุ่มของบิตคราวละหนึ่งบิตเรียงลำดับเรื่อยไปจนสิ้นสุดการสื่อสาร การสื่อสารแบบนี้จะมีข้อแตกต่างจากการสื่อสารแบบขนานเป็นอย่างมาก เนื่องจากข้อมูลมีการโอนย้ายมาพร้อมกัน จึงมีความจำเป็นต้องใช้จำนวนเส้นสัญญาณมากขึ้นตามจำนวนบิตของข้อมูลด้วย ในขณะที่การสื่อสารแบบอนุกรมนั้นต้องการเส้นสัญญาณเพียงสองหรือสามเส้นเท่านั้น ดังนั้นการสื่อสารแบบขนานจึงไม่เหมาะสมกับการสื่อสารกับอุปกรณ์ภายนอกเป็นระยะทางไกลๆ เพราะจะทำให้สิ้นเปลืองค่าใช้จ่ายมาก

2.6.2 ความเร็วของการสื่อสารข้อมูลอนุกรม

เนื่องจากการสื่อสารแบบอนุกรมเป็นการรับส่งข้อมูลในลักษณะกลุ่มบิตข้อมูล (bit stream) ดังนั้นจึงต้องให้ความสนใจในการพิจารณาถึงเรื่องอัตราความเร็วในการรับส่งบิตเหล่านี้เป็นลำดับแรก โดยทั่วไปมักจะระบุกันในหน่วยของจำนวนบิตข้อมูลในเวลาหนึ่งวินาทีเรียกว่า อัตราบอด ตามค่ามาตรฐานเหล่านี้ได้แก่ 110,150,300,1200,2400,4800,9600 และ 19200 บอด

ข้อมูลทั้ง 8 บิตนี้หากว่าถูกส่งออกมาด้วยอัตรา 2400 บอด จะใช้เวลาในการส่งข้อมูลหนึ่งบิต เท่ากับ 1/2400 หรือ 416 μS และเวลาในการส่งข้อมูลทั้ง 8 บิตมีค่าเท่ากับ 8×416 หรือ 3328 μS

2.6.3 รูปแบบของการส่งข้อมูลอนุกรม

การสื่อสารอนุกรมแบบอะซิงโครนัส จะใช้การแปลงข้อมูลขนานให้เป็นอนุกรมแล้วเพิ่มบิตบางอย่างร่วมไปกับการส่งข้อมูลจริงซึ่งได้แก่

2.6.3.1 บิตเริ่มต้น (start bit)

บิตเริ่มต้นมีหน้าที่สำหรับการบ่งบอกให้ทราบถึงตำแหน่งจุดเริ่มต้นก่อนบิตข้อมูล ตามปกติแล้วค่าของบิตเริ่มต้นจะเป็นระดับลอจิก ต่ำ

2.6.3.2 บิตแสดงภาวะความเป็นเลขคู่หรือเลขคี่ (parity bit)

บิตนี้มีหน้าที่เพื่อการตรวจสอบความถูกต้องของข้อมูล โดยทั่วไปมักเรียกว่า "บิตพาริตี้" และจะนำไปต่อท้ายบิตข้อมูล ค่าของบิตนี้ขึ้นอยู่กับจำนวนค่าของบิตที่เป็น 1 ซึ่งจะเป็นไปได้สองลักษณะคือ พาริตี้คู่ (even parity) หรือ พาริตี้คี่ (odd parity) ตัวอย่างเช่น ระบบที่ติดต่อกันโดยระบุว่าจะใช้พาริตี้คู่ ทางด้านส่งจะนำค่าข้อมูลที่จะส่งมาพิจารณาหาจำนวนของบิตที่มีค่าเป็น 1 ถ้าจำนวนบิตที่มีค่าเป็น 1 เป็นเลขจำนวนคู่อยู่แล้ว ค่าของบิตพาริตี้จะมีค่าเป็น 0 แต่หากว่าจำนวนบิตที่มีค่าเป็น 1 เป็นเลขจำนวนคี่ ค่าของบิตพาริตี้จะมีค่าเป็น 1 การพิจารณาทางด้านรับเป็นการตรวจสอบจำนวนบิตที่มีค่าเป็น 1 ของข้อมูลที่ได้รับมาทั้งหมดรวมทั้งบิตพาริตี้ ถ้า

เป็นเลขจำนวนคู่แสดงว่าข้อมูลที่ได้รับมาถูกต้อง แต่ถ้าหากไม่เป็นจำนวนคู่แสดงว่าเกิดการผิดพลาดของข้อมูล เป็นต้น

2.6.3.3 บิตสุดท้าย (stop bit)

บิตสุดท้ายเป็นบิตที่เพิ่มขึ้นเพื่อระบุถึงขอบเขตของการสิ้นสุดของกลุ่มบิตข้อมูล บิตสุดท้ายนี้สามารถโปรแกรมบิตได้คือ 1 บิต 1 1/2 บิต และ 2 บิต ดังนั้นกรณีของการส่งข้อมูล 8 บิตหากข้อมูลถูกส่งออกไปด้วยอัตราเร็ว 2400 บอด เวลาโดยรวมในการส่งข้อมูลหนึ่งไบต์จะมีค่าเป็น 12×416 μs หรือ 4.99 ms

2.6.4 การส่งข้อมูลอนุกรมของ 8051

พอร์ตอนุกรมของ8051 มีโครงสร้างการทำงานในแบบที่เรียกว่า ฟูลดูเพล็กซ์ (full duplex) ในการรับและการส่งข้อมูลอนุกรมได้ในเวลาเดียวกัน โดยด้านวงจรของตัวส่งจะมีข้อมูลออกไปยังพอร์ตอนุกรมทางขาสัญญาณ TxD (พอร์ท 3.1) ส่วนวงจรด้านตัวรับประกอบด้วย SBUF เช่นเดียวกับสัญญาณข้อมูลอนุกรมที่รับเข้ามาทางขาสัญญาณ RxD พอร์ตอนุกรมของ 8051 สามารถโปรแกรมการทำงานได้หลายโหมดด้วยกันโดยเลือกที่บิต SM0 และ SM1 ซึ่งอยู่ในรีจิสเตอร์ควบคุม SCON การทำงานทั้ง 4 โหมด ของพอร์ตอนุกรมมีดังนี้

2.6.4.1 โหมด 0 : ใช้รับส่งข้อมูล 8 บิตโดยการส่งจะเลื่อนออกทีละบิต โดยส่งบิต D_0 ออกไปก่อนทางขา RxD และไม่มีการส่งบิตเริ่มต้นแต่จะส่ง shift clockทางขา TxD ความเร็ว 1/12 เท่าของ CPU CLOCK

2.6.4.2 โหมด 1 : ใช้สำหรับการเชื่อมต่ออนุกรมแบบ UART (Universal Asynchronous Receiver/Transmitter) โดยการส่งแบบ 10 บิต 1 บิตเริ่มต้น และ 1 บิตสุดท้าย และสามารถเปลี่ยนแปลงอัตราความเร็วในการส่งข้อมูลได้ โดยขึ้นกับบิต SMOD ในPCON และอัตราโอเวอร์โฟลว์ของ TIMER1

2.6.4.3 โหมด 2 : ใช้สำหรับการเชื่อมต่ออนุกรมแบบ UART โดยการใช้ กลุ่มข้อมูลแบบ 11 บิตและกำหนดอัตราความเร็วในการส่งข้อมูลเท่ากับ 1/32 และ 1/64 ของCPU CLOCK โดยโปรแกรมที่บิต SMOD ในPCON

2.6.4.4 โหมด 3 : ใช้สำหรับการเชื่อมต่ออนุกรมแบบ UART โดยใช้กลุ่มข้อมูลแบบ 11 บิตและสามารถเปลี่ยนแปลงอัตราความเร็วในการส่งข้อมูลได้ โดยควบคุมที่บิต SMOD และอัตราโอเวอร์โฟลว์ของ TIMER1 นอกจากนี้โหมด 2 และ3 ยังมีการดำเนินการอีกแบบหนึ่ง โดยสามารถนำมาใช้ประโยชน์ในการสื่อสารข้อมูล แบบที่มีไมโครโปรเซสเซอร์หลายตัวทำงานร่วมกันได้ซึ่งมีชื่อเรียกว่า multiprocessor mode

สำหรับรีจิสเตอร์ SCON ซึ่งใช้ควบคุมการรับส่งข้อมูลอนุกรมมีรายละเอียดแสดงดังรูปที่ 2.17

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SCON.7	Serial Port mode specifier.(NOTE 1).
SM1	SCON.6	Serial Port mode specifier. (NOTE 1).
SM2	SCON.5	Enables the multiprocessor communication feature in mode 2 & 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0. (See Table 9).
REN	SCON.4	Set/Cleared by software to Enable/Disable reception.
TB8	SCON.3	The 9th bit that will be transmitted in modes 2 & 3. Set/Cleared by software.
RB8	SCON.2	In modes 2 & 3, is the 9th data bit that was received. In mode 1, if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.
TI	SCON.1	Transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes. Must be cleared by software.
RI	SCON.0	Receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or halfway through the stop bit time in the other modes (except see SM2). Must be cleared by software.

รูป 2.17 แสดงรายละเอียดของแต่ละบิตในรีจิสเตอร์ SCON

ตารางที่ 2.5 การโปรแกรมบิต SM0, SM1 เพื่อเลือกโหมดการทำงาน

SM0	SM1	โหมด	การทำงาน
0	0	0	ทำงานเป็น shift register อัตราความเร็วในการรับส่งข้อมูลเท่ากับ 1/12 ของความถี่ออสซิลเลเตอร์
0	1	1	8 bit UART อัตราเร็วในการรับหรือส่งข้อมูลกำหนดเองได้
1	0	2	9 bit UART อัตราเร็วในการรับหรือส่งข้อมูลเท่ากับ 1/32 หรือ 1/64 ของความถี่ออสซิลเลเตอร์โดยขึ้นกับบิต SMOD ใน PCON
1	1	3	9 bit UART อัตราเร็วในการรับหรือส่งข้อมูลนั้นสามารถกำหนดเองได้

- SM2 บิตเลือกแบบการทำงาน

  1 : เลือก Multiprocessor Mode ใช้กับโหมด 2, 3

  0 : เลือก Single Processor Mode ใช้ได้กับทุกโหมด

- REN บิตควบคุมให้รับหรือไม่รับข้อมูล

  1 : ให้รับข้อมูลได้

  0 : ห้ามรับข้อมูล

- TB8 (Transmitt bit D8) ข้อมูลบิตที่ 9 ที่จะส่งออกไปในโหมด 2 หรือ 3 ให้ใส่ในบิต TB8

- RB8 (Receivebit D8) ข้อมูลบิตที่ 9 ที่รับเข้ามาจะมาเก็บในบิตนี้ ข้อมูลบิตที่ 9 ก็คือค่าใน TB8 ทางด้านส่งนั่นเอง

- TI บิต TI จะเป็น 1 เมื่อสิ้นสุดการส่งข้อมูล 1 ไบต์

- RI บิต RI จะเป็น 1 เมื่อรับข้อมูลเสร็จ 1 ไบต์

2.6.5 การอินเตอร์รัปต์ของพอร์ตสื่อสารอนุกรม

เนื่องจากการรับหรือการส่งข้อมูลอนุกรมจะส่งทีละไบต์ 8051 จึงได้กำหนดให้บิตหรือแฟล็กสถานะที่จัดรวมในรีจิสเตอร์SCON เช่นแฟล็ก TI ซึ่งจะมีค่าเป็น 1 เมื่อข้อมูลได้ทำการส่งออกไปภายนอกเสร็จสิ้นแล้ว และแฟล็ก RI ซึ่งจะมีค่าเป็น 1 เพื่อให้รู้ว่าได้รับข้อมูลผ่านเข้ามาทางพอร์ตอนุกรมเสร็จแล้ว เมื่อแฟล็ก RI,TI นี้มีค่าเป็น 1 จะมีผลทำให้เกิดการอินเตอร์รัปต์ขึ้นดังนั้นภายในโปรแกรมรับหรือส่งข้อมูลจะต้องทำการตรวจสอบจากสภาวะของแฟล็กเหล่านี้เองว่าเป็นการส่งหรือการรับข้อมูล (vector ของ TI, RI อยู่ที่ 0023H)

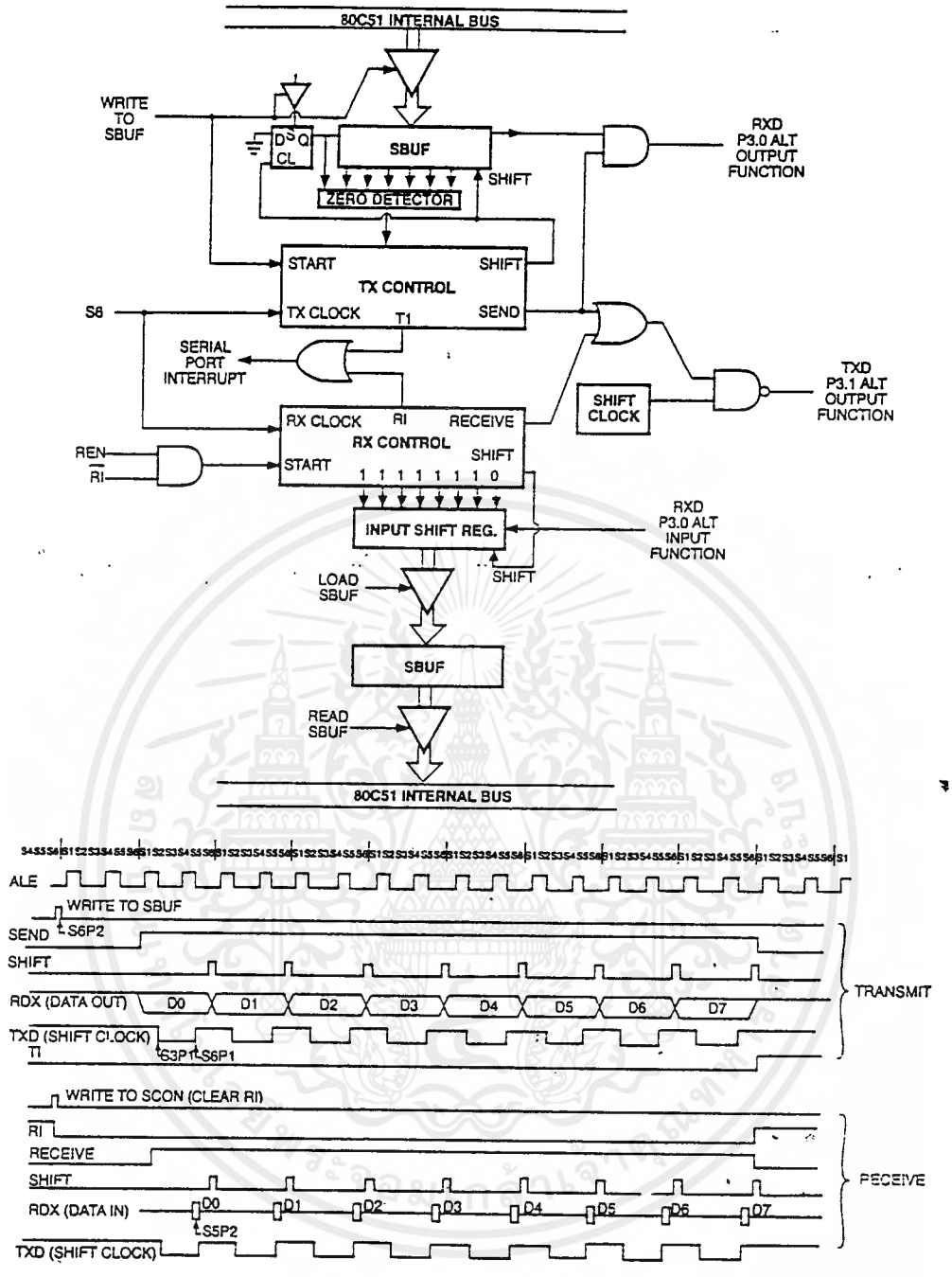
2.6.6 กระบวนการรับและส่งข้อมูลอนุกรมของ 8051

การส่งข้อมูลออกทางพอร์ตอนุกรมของ 8051 จะเริ่มขึ้นภายหลังเมื่อมีการเขียนข้อมูลลงใน SBUF ข้อมูลนี้จะถูกเลื่อนทีละบิตและส่งสัญญาณออกไปภายนอกโดยอัตโนมัติเมื่อข้อมูลเหล่านี้ได้ส่งออกครบถ้วนแล้วจะทำให้ค่าแฟล็กของ TI เป็น 1 เพื่อแจ้งให้ทราบว่าขณะนี้ SBUFว่างและพร้อมที่จะส่งข้อมูลไบต์ต่อไปแล้ว ในกรณีที่ใช้เขียนข้อมูลใหม่ลงในรีจิสเตอร์ SBUF โดยไม่รอให้แฟล็กTI มีค่าเป็น1 ก่อนจะมีผลทำให้ข้อมูลที่ส่งไปผิดพลาดได้

สำหรับการรับส่งข้อมูลจากพอร์ตอนุกรมจะต้องเริ่มต้นโดยการกำหนดเซ็ตค่าดังนี้ REN ให้มีค่าเป็น 1 ก่อนหลังจากนั้นเมื่อมีข้อมูลภายนอกถูกส่งเข้ามายัง 8051 ทีละบิตจนครบ และเมื่อบิตสุดท้ายถูกเลื่อนเข้ามาเรียบร้อยแล้วข้อมูลนั้นจะถูกย้ายมามาเก็บไว้ยังรีจิสเตอร์ SBUF และแฟล็ก RI ก็จะถูกเซ็ตให้มีค่าเป็น1 หลังจากนั้นจะเกิดการอินเตอร์รัปต์ขึ้น

2.6.6.1 การรับส่งข้อมูลอนุกรมโหมด 0

การทำงานของพอร์ตอนุกรมในโหมด 0 นี้ เป็นการรับและการส่งข้อมูลอนุกรมจำนวน 8 บิตโดยใช้เพียงขาสัญญาณ RxD เท่านั้น(ขานี้ใช้งาน 2 หน้าที่ใช้ส่งและรับข้อมูล) ส่วนขาสัญญาณ TxD จะนำไปใช้เพื่อเป็นขาสัญญาณนาฬิกาในการให้จังหวะการเลื่อนข้อมูลกับวงจรเลื่อนบิต สำหรับอัตราความเร็วจะถูกกำหนดไว้คงที่ที่ 1/12 ของความถี่ออสซิลเลเตอร์จากรูปที่ 2.18 แสดงให้เห็นถึงแผนภาพเวลาสัญญาณต่างๆ ในโหมด 0 เมื่อมีการรับและการส่งข้อมูล 1 ไบต์โดยสัญญาณนาฬิกาในการเลื่อนบิตนี้จะเกิดขึ้นภายในตัวของ 8051 เอง เนื่องจากโหมดนี้ไม่มีการส่ง start bit และ stop bitดังนั้นจึงมีความจำเป็นที่จะต้องส่งสัญญาณ shift clock ออกไปเพื่อใช้ synchronize ระหว่างฝ่ายรับและฝ่ายส่งโดยจะใช้ขา TxD ส่วนการรับข้อมูลจะรับข้อมูลเข้าทางขา RxD และรับ shift clock เข้าทางขา TxD ถ้าความถี่ออสซิลเลเตอร์มีค่าเท่ากับ 132 MHz ก็จะส่งได้ถึง 1 ล้านบอด ซึ่งโหมด 0 เป็นโหมดที่ส่งข้อมูลได้เร็วที่สุด รายละเอียดผังเวลาในการรับส่งดังแสดงในรูป 2.18

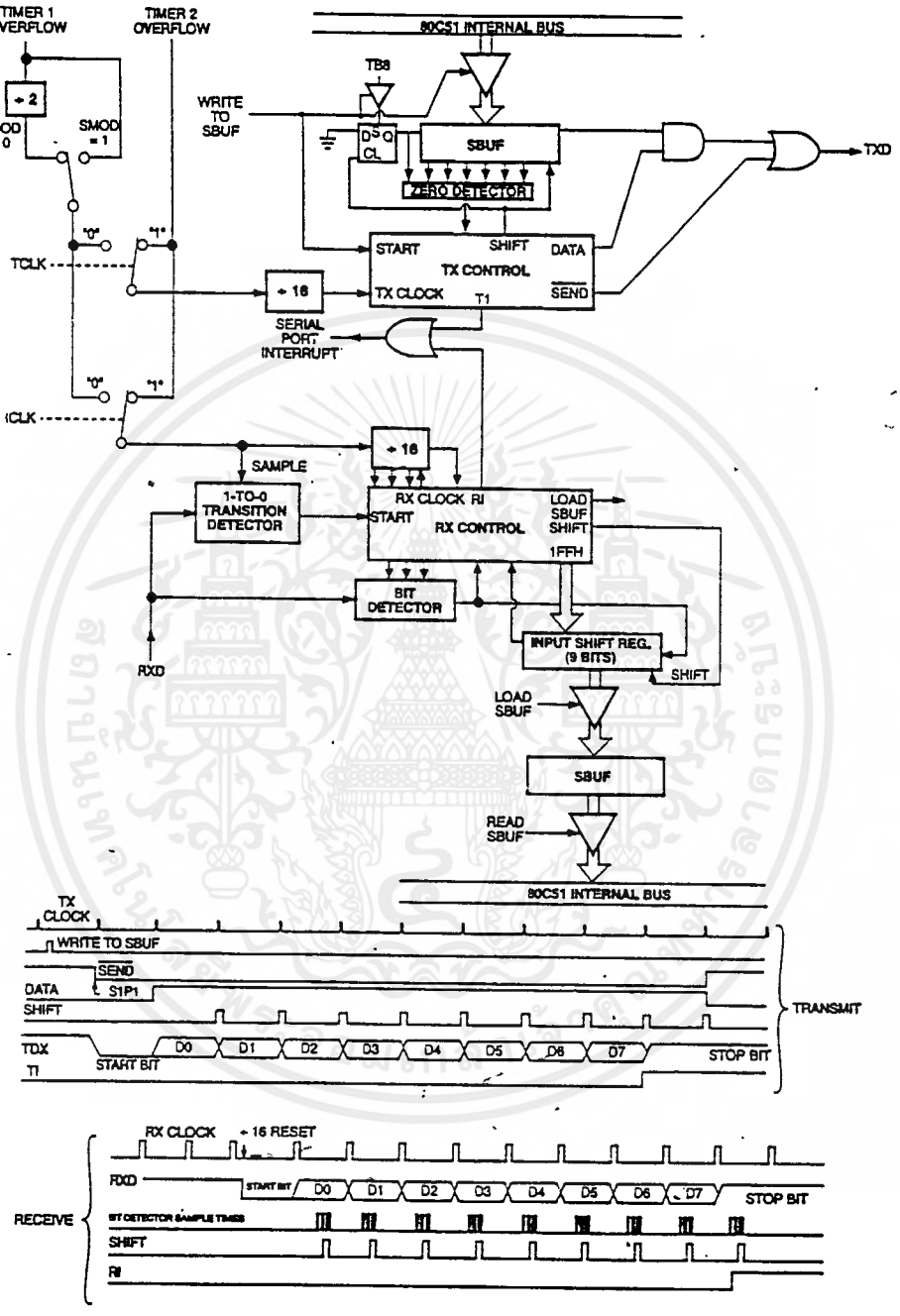


รูปที่ 2.18 แสดงผังการทำงานในโหมด 0

2.6.6.2 การรับส่งข้อมูลอนุกรมโหมด 1

การทำงานในโหมด 1 เป็นการสื่อสารข้อมูลอนุกรมจำนวน 10 บิต ประกอบด้วยบิตเริ่มต้น1 บิต ข้อมูล จำนวน 8 บิต และบิตสุดท้ายอีก 1 บิตดังแสดงในรูป 2.19 โดยข้อมูลจะถูกส่งออกทางขา TxD และรับเข้ามาทางขาสัญญาณ RxD ในส่วนของข้อมูล 8 บิต ที่ได้รับหรือการส่งออก จะเป็นบิตนัยสำคัญต่ำเป็นลำดับแรก

ส่วนทางฝ่ายรับค่าของ stop bit จะส่งเข้ามาจัดเก็บไว้ในบิต RB8ภายในรีจิสเตอร์ SCON สำหรับอัตราเร็วในการส่งข้อมูลของโหมด ๆ นั้นสามารถกำหนดเลือกได้จาก Timer 1ผังเวลาและการทำงานดังแสดงในรูปที่ 2.19



รูป 2.19 แสดงผังการทำงานในโหมด 1

ดังได้กล่าวแล้วว่าการส่งข้อมูลอนุกรมโหมด 1 สามารถเปลี่ยนแปลงความเร็วได้โดยการใช้ Timer 1 ทำหน้าที่เป็นตัวกำเนิดอัตราการส่งข้อมูลและใช้แฟล็กที่เกิดขึ้นจากการโอเวอร์โฟลว์ของ Timer 1 โดยโปรแกรม Timer 1 ทำงานใน 8-bit automatic reload โดยที่

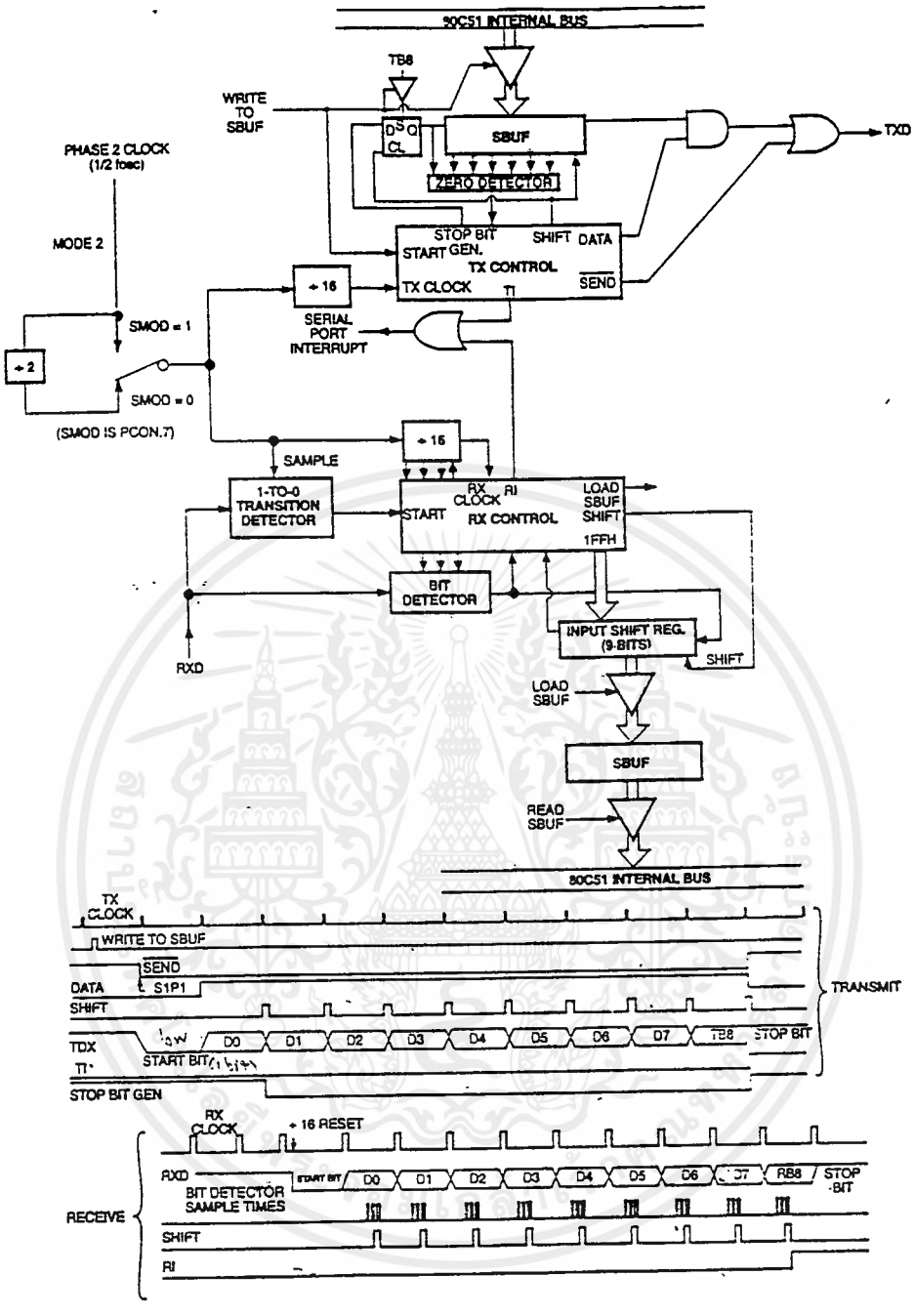
$$\text{ความถี่อัตราบอด} = \frac{(2)^{SMOD}}{(32)} \times \textit{Overflow Rate of Timer 1} \qquad (2.5)$$

$$\text{หรือความถี่อัตราบอด} = \left|\frac{(2)^{SMOD}}{(32)}\right| \times \left|\frac{f_{OSC}}{12 \times (256 - TH1)}\right| \qquad (2.6)$$

โดย SMOD เป็นค่าของบิตภายในรีจิสเตอร์ PCON (มีค่าเป็น 0 หรือ 1) ค่าภายในรีจิสเตอร์ TH1 จะถูกใช้เป็นค่าสำหรับโหลดซ้ำ

2.6.6.3 การรับส่งข้อมูลอนุกรมโหมด 2

โหมดนี้ใช้ทั้งหมด 11 บิต โดยแบ่งเป็น start bit , 9 data bit และ stop bit โดยบิตที่ 9 ผู้ใช้สามารถกำหนดค่าได้เองว่าจะส่งค่าอะไรออกไป โดยจะต้องนำไปใส่ไว้ในบิต TB8 ในรีจิสเตอร์ SCON ส่วนมากผู้ใช้มักจะนำบิตนี้มาใช้เป็น Parity bit โดยโหลดค่ามาจาก Parity flagใน PSW ส่วนทางด้านรับบิตที่9 จะถูกนำมาเก็บไว้ใน RB8 อัตราความเร็วในการส่งรับข้อมูลขึ้นกับความถี่ออสซิลเลเตอร์ของCPU และค่า SMODซึ่งอยู่ในบิตที่ 7 ใน PCON โดยผังเวลาและการทำงานดังแสดงในรูปที่ 2.20



รูปที่ 2.20 แสดงผังการทำงานในโหมด 2

$$\text{โดยที่ในโหมด2 มีค่าบอดเรท} = \frac{\left(2^{SMOD}\right) \times f_{OSC}}{64} \qquad (2.7)$$

ถ้า CPU run ที่12 MHz และ SMODมีค่าต่าง ๆ ดังนี้

$$\text{เมื่อ SMOD} = 0 \text{ จะได้} = \frac{2^0(12)(10^6)}{64} = 187{,}500 \text{ BAUD}$$

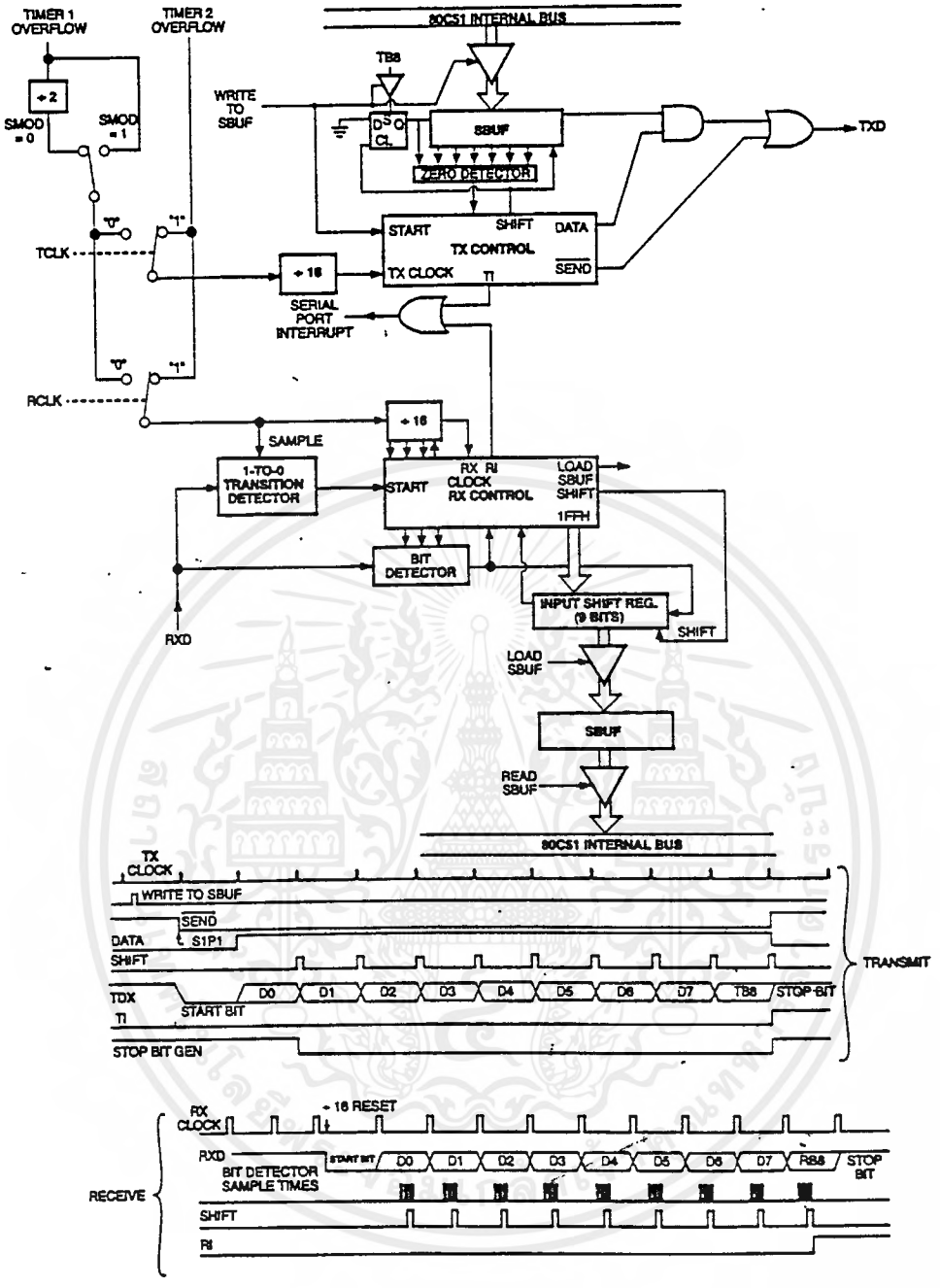
$$\text{เมื่อ SMOD} = 1 \text{ จะได้} = \frac{2^1(12)(10^6)}{64} = 375{,}000 \text{ BAUD}$$

2.6.6.4 การรับส่งข้อมูลอนุกรมโหมด 3

เหมือนกับโหมด2ทุกอย่างยกเว้นความเร็วในการรับส่งข้อมูลจะขึ้นกับอัตราโอเวอร์โฟลว์ของ Timer1 หรือ Timer2 โดยมีผังการทำงาน ดังรูป 2.21 และการเปลี่ยนแปลงความเร็วคำนวณจากสูตรดังนี้

เมื่อกำหนดให้ timer1 ทำงานในโหมด 2 Auto reload

$$\textbf{MODE 1,3} = \frac{(2^{SMOD})(f_{OSC})}{(32)12[256 - TH1]} \textbf{ BAUD} \qquad (2.8)$$



รูปที่ 2.21 แสดงผังการทำงานในโหมด 3

2.7 สเต็ปปิ้งมอเตอร์ (stepping motor)

2.7.1 โครงสร้างและการทำงานของสเต็ปปิ้งมอเตอร์

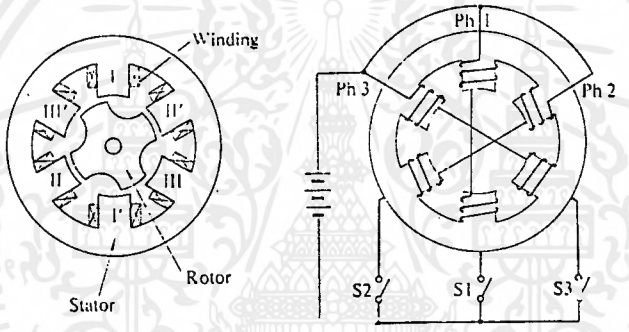
สเต็ปปิ้งมอเตอร์เป็นมอเตอร์ที่มีความแตกต่างจากมอเตอร์ทั่วๆ ไปโดยเมื่อป้อนกำลังไฟฟ้าให้กับมันๆจะหมุนเพียงเล็กน้อยตามเส้นรอบวงและหยุด ซึ่งต่างจากมอเตอร์ทั่วๆ ไปที่จะหมุนทันทีและตลอดเวลา

สเต็ปปิ้งมอเตอร์สามารถจะกำหนดตำแหน่งของการหมุนด้วยตัวเลขได้อย่างละเอียดโดยการใช้คอมพิวเตอร์เป็นตัวกำหนดและจัดเก็บตัวเลขเหล่านั้นไว้

สเต็ปปิ้งมอเตอร์สามารถใช้งานในระบบเปิด (open loop system) นั่นคือมันทำงานได้โดยไม่ต้องการการป้อนกลับ (feedback) แต่ทุกวิธีที่ต้องการกำหนดตำแหน่งได้อย่างถูกต้องจำเป็นที่จะต้องการการป้อนกลับไปยังระบบให้รับรู้และอะไรจะเป็นตัวบอกได้ว่าตำแหน่งถูกต้องแล้วหรือเกิดการผิดพลาด

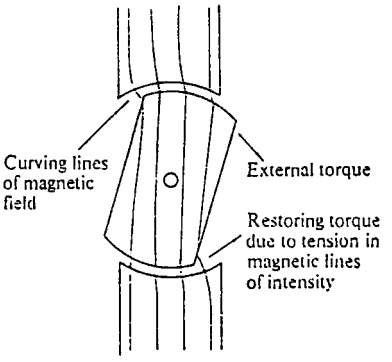
วิธีหนึ่งที่ใช้กันโดยทั่วไปกับสเต็ปปิ้งมอเตอร์ก็คือการใช้สวิทช์ติดตั้งไว้ที่ตำแหน่งที่ต้องการตรวจจับ (limit switch) เมื่อสเต็ปปิ้งมอเตอร์เริ่มหมุนและหมุนจนกระทั่งถึงตำแหน่งของสวิทช์ตรวจจับสัญญาณก็จะถูกป้อนกลับเข้าสู่ระบบและทราบการทำงานของสเต็ปปิ้งมอเตอร์ได้ตลอดเวลา ซึ่งโดยปกติในวงจรคอนโทรลเลอร์จะมีการกำหนดจุดอ้างอิง (reference point) ไว้ด้วย เพื่อให้เริ่มต้นทำงานและอ้างอิงตำแหน่งได้อย่างถูกต้อง

ภายในสเต็ปปิ้งมอเตอร์จะประกอบไปด้วย สเตเตอร์ , โรเตอร์ และขดลวดประกอบเข้าด้วยกันดังแสดงในรูปที่ 2.22 (สมมติเป็นมอเตอร์แบบ 3 เฟส)



รูปที่ 2.22 ภาพหน้าตัดของสเต็ปปิ้งมอเตอร์แบบ 3 เฟส และการจัดเรียงตัวของขดลวด

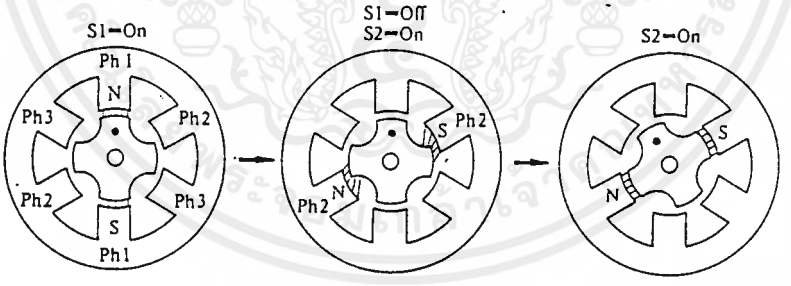
เนื่องจากสเต็ปปิ้งมอเตอร์นี้ โรเตอร์เป็นเหล็กอ่อนซึ่งมีคุณสมบัติที่จะพยายามปรับตัวเองให้อยู่ในแนวที่เส้นแรงแม่เหล็กผ่านมากที่สุดดังในรูปที่ 2.23 เมื่อเกิดเส้นแรงแม่เหล็กขึ้นที่สเตเตอร์ตัดผ่านโรเตอร์ตัวโรเตอร์ก็จะพยายามปรับตัวเองให้เส้นแรงแม่เหล็กตัดผ่านตัวเองมากที่สุดโดยการหมุนตัวเอง ทำให้เกิดมุมของการหมุนขึ้น และมอเตอร์จะหยุดหมุนเมื่อเส้นแรงแม่เหล็กที่ตัดผ่านตัวมันถึงจุดที่มากที่สุด



รูปที่ 2.23 เส้นแรงแม่เหล็กซึ่งทำให้เกิดแรงบิด

การทำให้สเต็ปปิ้งมอเตอร์หมุนก็ทำได้โดยอาศัยหลักการนี้ แต่ต้องให้เส้นแรงแม่เหล็กเกิดขึ้นโดยการรับช่วงต่อกันไปเรื่อยๆ ดังรูปที่ 2.24 ซึ่งแสดงถึงการหมุนของมอเตอร์ โดยทิศทางขึ้นอยู่กับการขับกระแสเข้าขดลวดว่าจะให้ไปทางใด และเมื่อต้องการให้มอเตอร์หยุดก็หยุดการขับโรเตอร์ มอเตอร์ก็จะหยุด ณ ตำแหน่งสุดท้ายที่มีการขับที่สเตเตอร์ ดังนั้นเราจึงสามารถรู้ตำแหน่งของมอเตอร์ได้โดยการนับจำนวนพัลส์ที่ ป้อนให้มอเตอร์โดยใช้สมการดังนี้

$$Different\ degree = Degree\ per\ step \times Numbers\ of\ input\ pulse \qquad (2.9)$$



รูปที่ 2.24 แสดงการเคลื่อนที่ทีละสเต็ปเมื่อมีการกระตุ้นจากเฟส 1 ไปยังเฟส 2

2.7.2 ชนิดของสเต็ปปิ้งมอเตอร์

สเต็ปปิ้งมอเตอร์แบ่งตามพื้นฐานได้เป็น 3 ชนิดคือ วาริเอเบิลรีลักแตนซ์ (variable reluctance : VR) , เพอร์มาเนนต์แมกเน็ต (permanent magnet : PM) และแบบ ไฮบริด (hybrid)

ชนิดวาริเอเบิลรีลักแตนซ์มีโครงสร้างของโรเตอร์แบบมัลติทูธ (multi-tooth) ทำจากเหล็กอ่อน เราจะทราบได้ว่าเป็นมอเตอร์ชนิดนี้โดยการทดสอบได้ง่ายมากคือใช้นิ้วหมุนเพลาของมอเตอร์และสังเกตมอเตอร์ชนิด

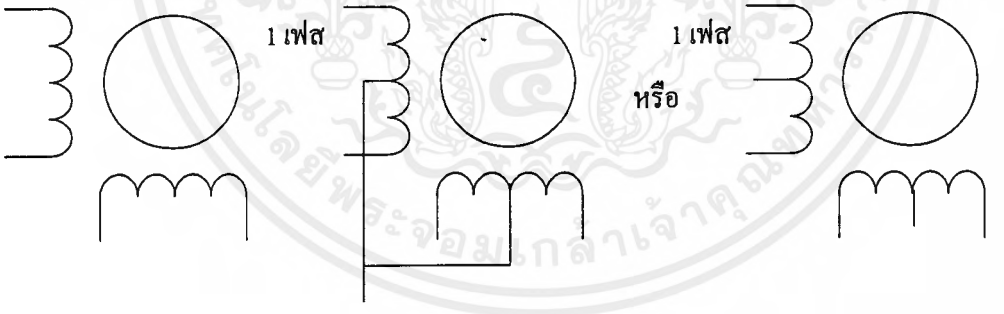
นี้ที่โรเตอร์จะไม่เกิดปรากฏการณ์ทางแม่เหล็ก (magnetism) มันจึงหมุนได้ตลอดไม่ติดขัดแตกต่างจากชนิด PM และชนิดไฮบริดซึ่งมีสนามแม่เหล็กที่โรเตอร์เมื่อหมุนจะรู้สึกขัดๆ เหมือนเป็นฟันเฟือง สเต็ปปิ้งมอเตอร์ชนิดนี้มีจุดด้อยในเรื่องของความถูกต้องของตำแหน่งและทำงานได้ไม่ดีนักเมื่อมีสเต็ปของการหมุนสูง

ชนิดเพอร์มาเนนต์แมกเน็ตมีโครงสร้างของโรเตอร์แบบเรียบไม่มีซี่ขั้วแม่เหล็กและบนโรเตอร์จะเป็นแบบแม่เหล็กถาวร การควบคุมทำได้โดยป้อนกระแสกระตุ้นที่ขดลวดบนสเตเตอร์ เช่นถ้าเป็นสเตเตอร์แบบ 4 เฟสจะมีขั้วแม่เหล็กอยู่ 4 ขั้วซึ่งมีคอยล์พันอยู่แยกจากกัน ขั้วแม่เหล็กถาวรบนโรเตอร์จะถูกแรงดึงดูดจากขั้วแม่เหล็กบนสเตเตอร์เมื่อป้อนกระแสไฟฟ้าเข้าสู่ขดลวด และโรเตอร์จะอยู่คงที่ที่ขั้วแม่เหล็กบนสเตเตอร์นั้นถึงแม้จะไม่ป้อนกระแสไฟฟ้าอีกต่อไป ทำให้เกิดเป็นแรงยึดหน่วงขึ้น สเต็ปปิ้งมอเตอร์ชนิดนี้มีข้อดีในเรื่องของความถูกต้องของตำแหน่งและความเร็วมากขึ้นเมื่อเปรียบเทียบกับชนิดอื่น

ชนิดไฮบริดเป็นชนิดที่นิยมใช้งานกันมากที่สุด โดยเฉพาะนำมาใช้กับงานอย่างมากในอุปกรณ์ที่ใช้ในเครื่องคอมพิวเตอร์ ชนิดไฮบริดมีโครงสร้างภายในซึ่งได้จากการรวมเอาโครงสร้างของสเตเตอร์ชนิดวาริเอเบิลรีลักแตนซ์และโครงสร้างของโรเตอร์จากชนิดเพอร์มาเนนต์แมกเน็ตมาประกอบเข้าด้วยกันจึงทำให้เป็นมอเตอร์ชนิดที่มีแรงยึดหน่วงสูง , มีแรงบิดดึงและผลักได้ดีซึ่งมีความคงที่และทำงานได้ดีถึงแม้ว่าจะมีสเต็ปต่อรอบในการหมุนสูง

2.7.3 การพันขดลวดบนสเต็ปปิ้งมอเตอร์

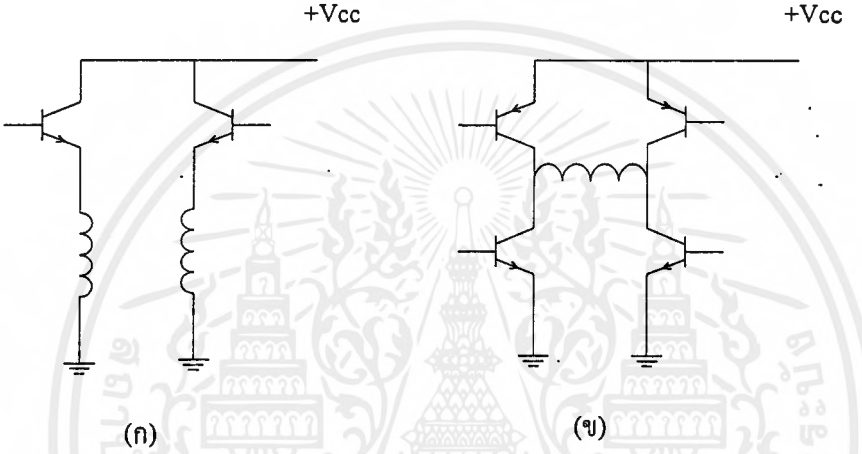
การพันขดลวดหรือคอยล์บนสเต็ปปิ้งมอเตอร์มีอยู่ 2 วิธีคือ แบบไบโพลาร์ (bipolar) และแบบยูนิโพลาร์ (unipolar) ดังแสดงในรูปที่ 2.25



รูปที่ 2.25 การพันขดลวดบนสเตเตอร์ของสเต็ปปิ้งมอเตอร์ โดยทางด้านซ้ายเป็นแบบไบโพลาร์และที่เหลือเป็นแบบยูนิโพล่าร์ ซึ่งมีทั้งแบบ 5 สาย และ 6 สาย 4 เฟส

สเต็ปปิ้งมอเตอร์แบบไบโพลาร์มีการพันขดลวด 1 ขดบนแต่ละขั้วแม่เหล็กของสเตเตอร์ ขั้วแม่เหล็กที่เกิดขึ้นบนสเตเตอร์ถูกกำหนดโดยทิศทางของกระแสไฟฟ้า และสามารถทำให้เกิดขั้วแม่เหล็กในทิศทางตรงข้ามได้โดยการกลับทิศการไหลของกระแสไฟฟ้า ซึ่งการกำหนดทิศทางการไหลและการกลับทิศทางของกระแสไฟฟ้าทำได้โดยการใช้วงจรสวิทช์ซึ่งกลับขั้วไฟฟ้า

สำหรับยูนิโพลาร์จะมีการพันขดลวด 2 ขดบนแต่ละขั้วแม่เหล็กของสเตเตอร์ ซึ่งแต่ละขดจะทำให้เกิดขั้วแม่เหล็กในทิศทางตรงข้ามกัน การกลับขั้วแม่เหล็กเปลี่ยนไปมาทำได้โดยการสวิทช์ชิ่งกระแสไฟฟ้าจากขดลวดขดหนึ่งไปยังอีกขดหนึ่งแทนเท่านั้น โดยปกติขดลวดทั้ง 2 จะมีการเชื่อมต่อกันหรือมีจุดร่วมเพื่อลดจำนวนของสายไฟที่ต่อจากมอเตอร์ วงจรจ่ายกำลังไฟฟ้าของมอเตอร์แบบยูนิโพลาร์ทำได้ง่ายกว่าชนิดไบโพลาร์ เพราะมันต้องการเพียงสวิทช์ธรรมดาในการเปิดและปิดกำลังไฟฟ้าให้กับขดลวดบนสเตเตอร์ในทิศทางที่ต้องการให้หมุนได้ทันที รูปที่ 2.26 แสดงวงจรจ่ายกำลังไฟฟ้าซึ่งใช้ทรานซิสเตอร์ทำหน้าที่เป็นตัวสวิทช์ชิ่งให้กับสเต็ปปิ้งมอเตอร์ที่มีการพันขดลวดทั้ง 2 แบบ จะเห็นได้ว่าในแบบของยูนิโพลาร์เป็นวงจรที่ง่ายและไม่มีความซับซ้อนเลย



รูปที่ 2.26 แสดงวงจรจ่ายกำลังไฟฟ้าให้กับสเต็ปปิ้งมอเตอร์ทั้ง 2 แบบ
(ก) สำหรับชนิดยูนิโพลาร์ ซึ่งใช้ทรานซิสเตอร์สวิทช์เพียงตัวเดียวต่อ 1 คอยล์
(ข) สำหรับชนิดไบโพลาร์ ซึ่งต้องใช้ทรานซิสเตอร์สวิทช์ 4 ตัวต่อ 1 คอยล์

อย่างไรก็ตามการพันขดลวดแบบยูนิโพลาร์ก็มีจุดด้อยตรงที่การพันแบบนี้จะทำให้เกิดแรงบิดน้อยกว่าแบบไบโพลาร์เพราะจะมีเพียงครึ่งหนึ่งของขดลวดที่ถูกกระตุ้นให้ทำงานเท่านั้นในระยะเวลาหนึ่ง

การพิจารณาว่าสเต็ปปิ้งมอเตอร์ตัวใดมีการพันขดลวดแบบใดสังเกตได้ง่ายโดยถ้าเป็นแบบไบโพลาร์จะมีสายไฟต่อออกจากมอเตอร์เพียง 4 สาย และถ้าเป็นแบบยูนิโพลาร์จะมี 5 หรือ 6 สาย

2.7.4 การกระตุ้นและควบคุมการหมุนของสเต็ปปิ้งมอเตอร์

การกระตุ้นและควบคุมการหมุนของมอเตอร์ให้เคลื่อนที่ไปแต่ละสเต็ปทำได้โดยจ่ายกำลังไฟฟ้าไปยังขดลวดแต่ละขดบนสเตเตอร์ ซึ่งต้องป้อนเป็นแบบซีเควนเชี่ยลในรูปแบบที่ถูกต้องด้วยให้เรียงกันไปเรื่อยๆทางใดทางหนึ่ง ถ้าต้องการให้หมุนกลับก็กระตุ้นเฟสในทิศทางกลับกัน โดยสามารถแบ่งออกได้เป็น 3 รูปแบบคือ

2.7.4.1 การกระตุ้นเฟสเดียวที่เรียกว่า **single phase excitation**

2.7.4.2 การกระตุ้น 2 เฟสที่เรียกว่า **two phase excitation**

2.7.4.3 การกระตุ้นโดยใช้แบบ 1 และ 2 สลับกันเรียกว่าแบบ **one-two-phase excitation** หรือแบบ **half-step operation**

2.7.4.1 การกระตุ้นแบบเฟสเดียว (single phase excitation)

single phase excitation เป็นรูปแบบการกระตุ้นที่ง่ายที่สุด โดยทำการกระตุ้นขดลวดทีละขดในเวลาหนึ่งและเรียงถัดกันไป ดังเช่น ขดที่ 1, 2,3,4,1 หรือ 1,4,3,2,1 ขึ้นอยู่กับทิศทางที่ต้องการให้หมุน ดังนั้นจึงมีขดลวดเพียงขดเดียวในเวลาหนึ่งที่ถูกกระตุ้นเท่านั้น วงจรกระตุ้นแบบเวฟจึงมีราคาถูกและง่าย ขั้นตอนการทำงานต่างๆแสดงดังในตารางที่ 2.6

ตารางที่ 2.6 แสดงขั้นตอนการกระตุ้นขดลวดแต่ละเฟสแบบ single phase excitation

สเต็ปที่	เฟสที่ 1	เฟสที่ 2	เฟสที่ 3	เฟสที่ 4
1	ทำงาน			
2		ทำงาน		
3			ทำงาน	
4				ทำงาน

2.7.4.2 การกระตุ้นแบบ 2 เฟส (two phase excitation)

เป็นการกระตุ้นอีกแบบหนึ่งซึ่งคล้ายกับแบบแรก แต่การกระตุ้นแบบนี้จะทำการกระตุ้นโดยจ่ายกำลังไฟฟ้าไปที่ขดลวด 2 ขดที่อยู่ใกล้กันในเวลาเดียวกันและเรียงถัดกันไปเช่นเดียวกับแบบแรก คือ ขดลวดที่ถูกกระตุ้น 12,23,34,41,12 หรือ 14,43,32,21,14 ขึ้นอยู่กับทิศทางการหมุน การเพิ่มจำนวนของขดลวดที่ถูกกระตุ้นนี้ทำให้เพิ่มแรงบิดได้มากกว่าแบบแรก โรเตอร์จะเคลื่อนที่ด้วยแรงดึงอย่างเต็มแรงจาก 2 ขดลวดที่ถูกกระตุ้นพร้อมกัน และต่อไปด้วยแรงดึงจากอีก 2 ขดลวดถัดไป สำหรับข้อเสียก็คือการกระตุ้นแบบนี้ต้องใช้แหล่งจ่ายกำลังไฟฟ้ามากขึ้น ขั้นตอนการทำงานต่างๆ แสดงดังในตารางที่ 2.7

ตารางที่ 2.7 แสดงขั้นตอนการกระตุ้นขดลวดแต่ละเฟสแบบ two-phase excitation

สเต็ปที่	เฟสที่ 1	เฟสที่ 2	เฟสที่ 3	เฟสที่ 4
1	ทำงาน	ทำงาน		
2		ทำงาน	ทำงาน	
3			ทำงาน	ทำงาน
4	ทำงาน			ทำงาน

2.7.4.3 การกระตุ้นแบบ half-step operation

แบบครึ่งสเต็ปนี้เป็นรูปแบบที่เกิดจากการผสมผสานระหว่างการกระตุ้นแบบแรกและแบบ 2 เพื่อเพิ่มจำนวนสเต็ปต่อรอบอีกเท่าตัวหนึ่ง ในระบบนี้จะทำการกระตุ้นขดลวดเรียงกันไปเป็นลำดับดังนี้ ขดลวดที่ถูกกระตุ้น 1,12,2,23,3,34,4,41,1 หรือในการหมุนอีกทิศทางหนึ่งจะได้เป็น 1,14,4,43,3,32,2,21,1 แรงบิดที่ได้จากการกระตุ้นแบบนี้จะเพิ่มมากขึ้นอีก เพราะช่วงสเต็ปมีระยะสั้นลงและแต่ละสเต็ปเกิดแรงดึงจากขดลวด 2 ขดที่ถูกกระตุ้นพร้อมกัน ความถูกต้องของตำแหน่งมีเพิ่มมากขึ้น แต่ต้องพึงระวังไว้อีกประการหนึ่งว่าเมื่อกระตุ้นให้ทำ งานในรูปแบบนี้จะต้องทำการหมุนถึง 2 สเต็ปจึงจะได้เท่ากับ 1 สเต็ปเต็มเหมือนกับในการควบคุม 2 แบบแรก สำหรับแหล่งจ่ายกำลังไฟฟ้าต้องใช้เทียบเท่ากับแบบ 2 เฟสจึงจะเพียงพอ ขั้นตอนการทำงานต่างๆ แสดงดังในตารางที่ 2.8

ตารางที่ 2.8 แสดงขั้นตอนการกระตุ้นขดลวดแต่ละเฟสแบบ half-step operation

สเต็ปที่	เฟสที่ 1	เฟสที่ 2	เฟสที่ 3	เฟสที่ 4
1	ทำงาน			
2	ทำงาน	ทำงาน		
3		ทำงาน		
4		ทำงาน	ทำงาน	
5			ทำงาน	
6			ทำงาน	ทำงาน
7				ทำงาน
8	ทำงาน			ทำงาน

2.7.5 การเกิดออสซิลเลตของสเต็ปปิ้งมอเตอร์

ความเร็วของการหมุนสเต็ปปิ้งมอเตอร์เป็นลิเนียร์กับความถี่ที่ป้อนให้สเต็ปปิ้งมอเตอร์ แต่เมื่อเราป้อนความถี่ให้เพิ่มขึ้นเรื่อยๆ จนถึงความถี่ค่าหนึ่งสเต็ปปิ้งมอเตอร์ก็จะหยุดหมุน เนื่องจากการที่โรเตอร์หมุนตามฟลักซ์แม่เหล็กไม่ทัน เราเรียกว่า มอเตอร์เกิดการออสซิลเลต ซึ่งมอเตอร์แต่ละตัวจะออสซิลเลตที่ความถี่ต่างกันไป โดยทั่วไปจะออสซิลเลตที่ความถี่ประมาณ 500 Hz ในการกระตุ้นแบบ 1 หรือ 2 เฟส และจะออสซิลเลตที่ความถี่ประมาณ 1 kHz เมื่อขับแบบครึ่งสเต็ป

แต่เมื่อลดความถี่ให้ต่ำลงมอเตอร์จะไม่หมุนทันทีและเมื่อลดความถี่ลงจนถึงความถี่หนึ่งมอเตอร์จึงจะเริ่มหมุนอีกครั้ง นั่นก็คือ มอเตอร์มีฮิสเทอรีซิส ซึ่งมอเตอร์จะเริ่มหมุนที่ความถี่ประมาณ 200 Hz ในการขับแบบ 1 หรือ 2 เฟส และมีความถี่ประมาณ 150 Hz เมื่อขับแบบครึ่งสเต็ป

บทที่ 3
การออกแบบและการสร้าง

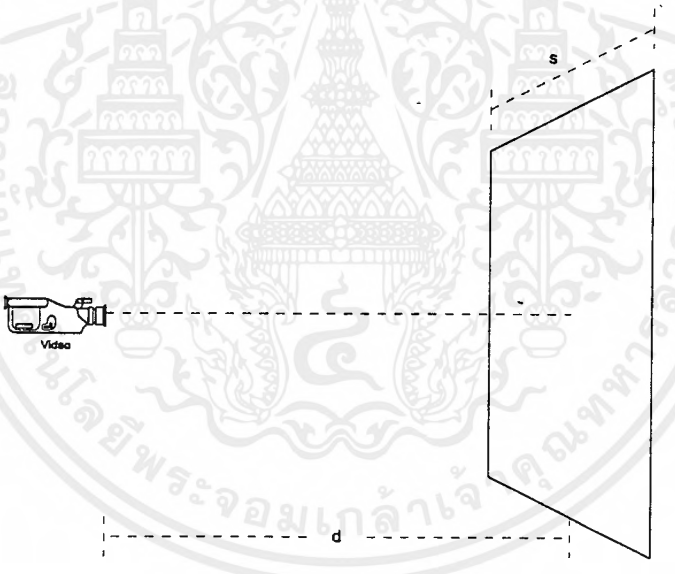
3.1 การออกแบบการทำงานของเมนูการติดตามวัตถุอัตโนมัติ

ในการที่จะออกแบบการทำงานของโปรแกรมเพื่อทำการควบคุมการหมุนของกล้องให้สามารถติดตามวัตถุได้นั้น เราจำเป็นที่จะต้องทราบถึงคุณลักษณะของชุดอุปกรณ์ที่ใช้ควบคุมการหมุนดังต่อไปนี้เสียก่อน

- อัตราการทดรอบของชุดเฟืองเท่ากับ 20:1
- ลักษณะการหมุนของสเต็ปปิ้งมอเตอร์ 7.5:1 (สเต็ปต่อองศา)

ต่อไปจะเป็นขั้นตอนการออกแบบโปรแกรมโดยจะสามารถแบ่งออกเป็นขั้นตอนต่างๆ ได้ดังต่อไปนี้

3.1.1 ในขั้นตอนแรกโปรแกรมจะแสดงกล่องโต้ตอบกับผู้ใช้ (dialog box) เพื่อให้ผู้ใช้ทำการป้อนระยะห่างระหว่างฉากกับตัวกล้อง (d) และขนาดของฉาก (s) ซึ่งจะใช้หน่วยการวัดเป็นเซนติเมตร ซึ่งค่าเหล่านี้จะถูกนำไปใช้เพื่อคำนวณหาจำนวนพัลส์ที่ป้อนให้กับสเต็ปปิ้งมอเตอร์ เมื่อวัตถุมีการเคลื่อนที



รูปที่ 3.1 แสดงถึงการจัดตั้งระบบติดตามวัตถุอัตโนมัติ

สมมติให้

ความกว้างของฉากเท่ากับ	s	cm.
ระยะห่างระหว่างฉากกับกล้อง	d	cm.
ขนาดภาพที่ปรากฏบนจอคอมพิวเตอร์	m	pixels

และจากการป้อนพัลส์ให้กับสเต็ปปิ้งมอเตอร์ 1 พัลส์ จะทำให้มอเตอร์หมุน 7.5 องศา ซึ่งจะทำให้ตัวกล้องหมุนไป 7.5/20 องศา หรือ

ระยะทางที่เกิดจากการหมุนของกล้อง 1 องศา เท่ากับ $\frac{2d\pi}{360} \times \frac{7.5}{20}$ cm.

ดังนั้นถ้าวัตถุมีการเคลื่อนที่ไป 1 cm. เราจะต้องป้อน $\frac{360 \times 20}{2d\pi \times 7.5}$ พัลส์

และจากระยะทางจริงต่อ 1 พิกเซล มีค่าเท่ากับ $\frac{s}{m}$ cm/pixel

นั่นคือ เราจะได้ขนาดพิกเซลต่อระยะทางจริงเท่ากับ

$$\text{1 pixel} = \frac{152.789s}{dm} \quad (3.1)$$

3.1.2 ในขั้นตอนที่สองจะเป็นการถ่ายเฟรมต้นแบบเพื่อเก็บเป็นเฟรมอ้างอิงในการเปรียบเทียบกับเฟรมต่อมา

3.1.3 ทำการถ่ายเฟรมต่อมาเพื่อที่จะนำไปเปรียบเทียบกับกับเฟรมที่ได้จากขั้นตอนที่สอง

3.1.4 ทำการเปรียบเทียบเฟรมที่ได้จากข้อ 3.1.2 และข้อ 3.1.3 โดยจะใช้วิธีนำค่าพิกเซลในแต่ละเฟรมมาทำการลบกัน โดยถ้าเป็นภาพเดียวกันจะได้ค่าจะได้ค่าของพิกเซลแต่ละพิกเซลเป็น 0 หมด แต่ถ้าตำแหน่งไหนของภาพมีความแตกต่างกันจะได้ผลลัพธ์ออกมาไม่เป็น 0

3.1.5 จากขั้นตอนที่ 3.1.4 ถ้าผลการเปรียบเทียบภาพเป็น 0 หมด แสดงว่าเฟรมที่ถ่ายมาทั้งสองเหมือนกันนั่นคือวัตถุไม่มีการเคลื่อนที่ ซึ่งจะต้องกลับไปทำงานในขั้นตอนที่ 3.1.3 แต่ถ้าผลการเปรียบเทียบภาพแล้วผลลัพธ์ที่ได้ไม่เป็น 0 หมด ก็จะทำการหาจุดเซนทรอยด์ของวัตถุโดยใช้สูตร

$$X_C = \int \frac{XdA}{A} \quad (3.2)$$

$$Y_C = \int \frac{YdA}{A} \quad (3.3)$$

แต่ในโครงงานนี้จะสนใจเฉพาะค่าตำแหน่งในแกน X เท่านั้น

3.1.6 พิจารณาค่า Xc ที่ได้จากขั้นตอนที่ 3.1.5 ว่าน้อยกว่าหรือมากกว่าจุดกึ่งกลางของภาพบนคอมพิวเตอร์เท่าไร ซึ่งเราก็จะได้ค่าผลต่างออกมาในรูปของขนาดพิกเซล

3.1.7 ทำการคำนวณจำนวนพัลส์ที่จะต้องส่งไปให้กับสเต็ปปิ้งมอเตอร์ โดยนำค่าที่ได้จากขั้นตอนที่ 3.1.6 มาคูณกันกับสูตรในขั้นตอนที่ 3.1.1

3.1.8 ทำการส่งค่าที่ได้จากขั้นตอนที่ 3.1.7 พร้อมกับทิศทางการหมุนของกล้องไปให้กับ MCS-51

3.1.9 รอรับการตอบรับจาก MCS-51 เพื่อกลับไปทำงานในขั้นตอนที่ 3.1.2 ต่อไป

สำหรับขั้นตอนการออกแบบทั้งหมดในส่วนของการประมวลผลภาพนั้นสามารถแสดงได้ดังโฟล์วชาร์ทที่ 3.1 , 3.2 และ 3.2 ตามลำดับ

3.2 การออกแบบการทำงานในส่วนควบคุมการหมุนของสเต็ปปิ้งมอเตอร์

ในส่วนควบคุมการหมุนของมอเตอร์จะอาศัยการควบคุมจากไมโครโปรเซสเซอร์ตระกูล MCS-51 ให้ทำการสร้างสัญญาณพัลส์ส่งออกทางพอร์ท1 ที่ตำแหน่ง 4 บิตล่างซึ่งทำการต่อกับเฟสทั้ง 4 เฟสของมอเตอร์ผ่านทางวงจรขับกระแส โดยมีการออกแบบขั้นตอนการทำงานของโปรแกรมเป็นลักษณะดังต่อไปนี้

3.2.1 ทำการกำหนดค่าเริ่มต้นให้แก่รีจิสเตอร์ภายในไมโครโปรเซสเซอร์รวมถึงกำหนดสถานะของพอร์ท เพื่อเป็นการจัดเตรียมช่องทางสำหรับรับส่งข้อมูลจากคอมพิวเตอร์ผ่านทาง RS-232 รวมถึงกำหนดการทำงานภายในไมโครโปรเซสเซอร์เอง อาทิเช่น โหมดการรับส่งข้อมูล บอดเรต เป็นต้น

3.2.2 รับข้อมูลซึ่งเป็นตัวกำหนดทิศทางในการหมุนของมอเตอร์ว่าจะให้หมุนไปทางซ้ายหรือทางขวา จำนวนองศาที่จะให้มอเตอร์หมุนเพื่อให้กล้องวีดีโอทำการติดตามวัตถุที่กำลังเคลื่อนที่

3.2.3 นำข้อมูลที่รับได้จากข้อที่ 3.2.2 มาทำการสร้างพัลส์แล้วส่งออกทางพอร์ท 1 ที่ 4 บิตล่าง เพื่อทำการหมุนมอเตอร์

3.2.4 ทำการตรวจสอบว่ามอเตอร์ได้หมุนไปในตำแหน่งที่ต้องการหรือยัง ถ้ายังก็ให้ทำการสร้างสัญญาณพัลส์เพื่อส่งออกให้กับมอเตอร์ต่อไป แต่ถ้าได้หมุนไปยังตำแหน่งที่ต้องการแล้วก็ให้ทำงานตามขั้นตอนที่ 3.2.5 ต่อไป

3.2.5 ทำการส่งข้อมูลตอบรับกลับไปยังคอมพิวเตอร์ว่าได้หมุนมอเตอร์ไปเรียบร้อยแล้ว และย้อนกลับไปรอรับข้อมูลตามขั้นตอน 3.2.2 ต่อไปเป็นเช่นนี้ไปเรื่อย ๆ

สำหรับโฟล์วชาร์ทในส่วนของการควบคุมมอเตอร์สามารถแสดงได้ดังรูปที่ 3.4 และ 3.5 ตามลำดับ

3.3 การกำหนดความถี่ของสัญญาณพัลส์

เนื่องจากสเต็ปปิ้งมอเตอร์แต่ละตัวจะมีการทำงานในช่วงความถี่ที่แตกต่างกันไป ดังนั้นการกำหนดความถี่ของสัญญาณพัลส์ที่จะป้อนให้กับมอเตอร์จึงเป็นสิ่งสำคัญ เนื่องจากถ้ามีการกำหนดความถี่ที่ไม่เหมาะสมแล้วมอเตอร์ก็จะไม่ทำการหมุนไปตามที่ต้องการ จากการทดลองสุ่มความถี่จะพบว่ามอเตอร์ที่ทำการทดลองจะหมุนในช่วงความถี่ประมาณ 120 ถึง 150 Hz ดังนั้นเราจึงทำการสร้างสัญญาณพัลส์ในช่วงความถี่นี้โดยทำการสร้างสัญญาณพัลส์ที่ความถี่ประมาณ 134 Hz ที่ตำแหน่งพอร์ท 1 ซึ่งได้มีการอาศัยการเขียนโปรแกรมย่อยเพื่อทำการหน่วงเวลา (delay) ค่าสถานะใน 4 บิตล่างของพอร์ท 1 ทำให้ได้เป็นสัญญาณพัลส์ที่มีความถี่ตามต้องการ

สำหรับโปรแกรมย่อยเพื่อทำการหน่วงเวลา ซึ่งเป็นส่วนหนึ่งของโปรแกรมควบคุมการหมุนของมอเตอร์นั้น จะมีโครงสร้างของโปรแกรมเป็นแบบวนลูปจำนวน 2 ลูป ดังนี้

```
        MOV     R0,# A
DELAY:  MOV     R1,# B
DEL:    NOP
        NOP
        DJNZ    R1,DEL
        DJNZ    R0,DELAY
        RET
```

โดยสูตรแสดงจำนวนเวลาที่หน่วงไปโดยโปรแกรมสามารถแสดงได้ดังนี้

$$\text{Delay Time} = 3 + A(5 + 2(B)) \quad \text{machine cycle} \qquad (3.4)$$

โดยเราทำการกำหนดให้ค่า A = 10 และค่า B = 83 แล้วทำการโหลดค่าทั้งสองนี้เข้าตัวนับ ดังนั้นค่า Delay Time จะเป็น

$$\text{Delay Time} = 3 + 10(5 + 2(83)) \quad \text{machine cycle}$$
$$= 1713 \quad \text{machine cycle}$$

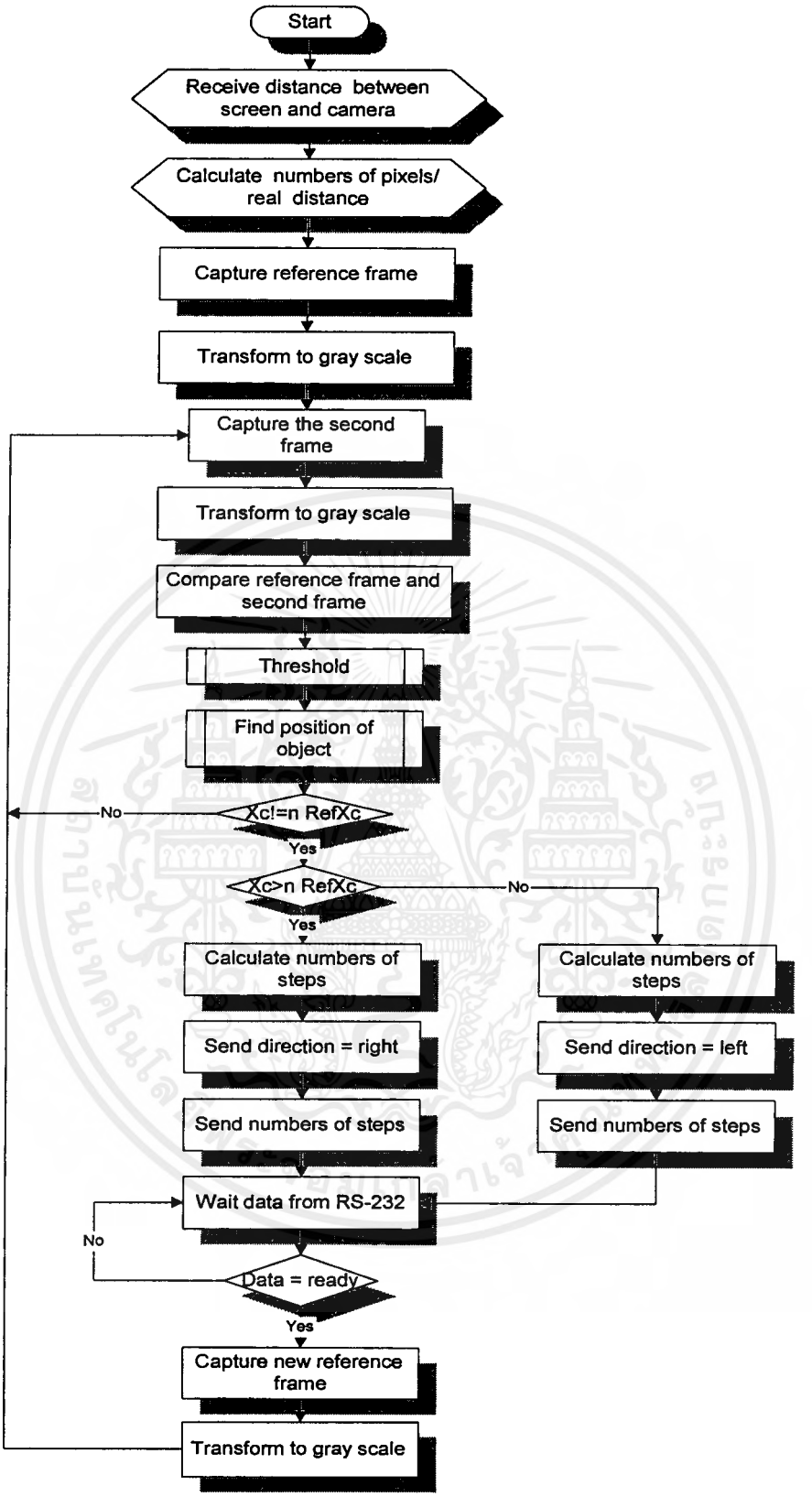
เนื่องจากไมโครโปรเซสเซอร์รันที่ความถี่ 11.059 MHz ดังนั้น 1 machine cycle จะกินเวลาเท่ากับ 1.09 μs เพราะฉะนั้นจะได้ว่าค่า Delay Time มีค่าเป็น

$$\text{Delay Time} = 1713 \times 1.09 \quad \mu s$$
$$= 1867.17 \quad \mu s$$

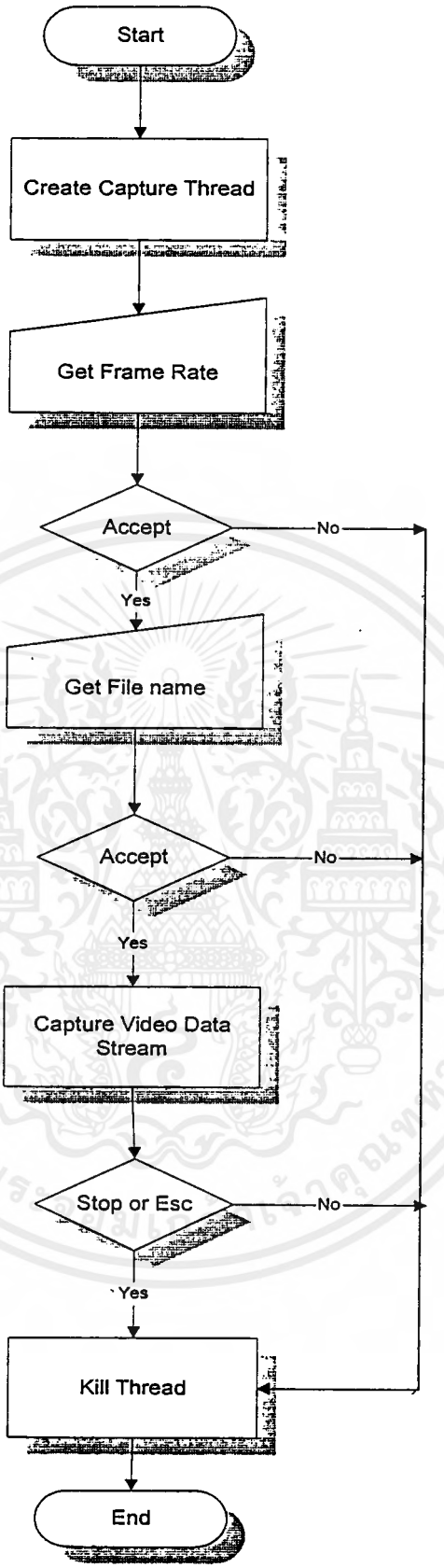
เนื่องจากค่า Delay Time ที่ได้นี้เป็น 1 ใน 4 ของคาบเวลาของสัญญาณดังนั้น

$$\text{คาบเวลาของสัญญาณ} = 4 \times 1867.17 \quad \mu s$$
$$= 7468.68 \quad \mu s$$
$$\text{หรือมีความถี่} \cong 134 \quad Hz$$

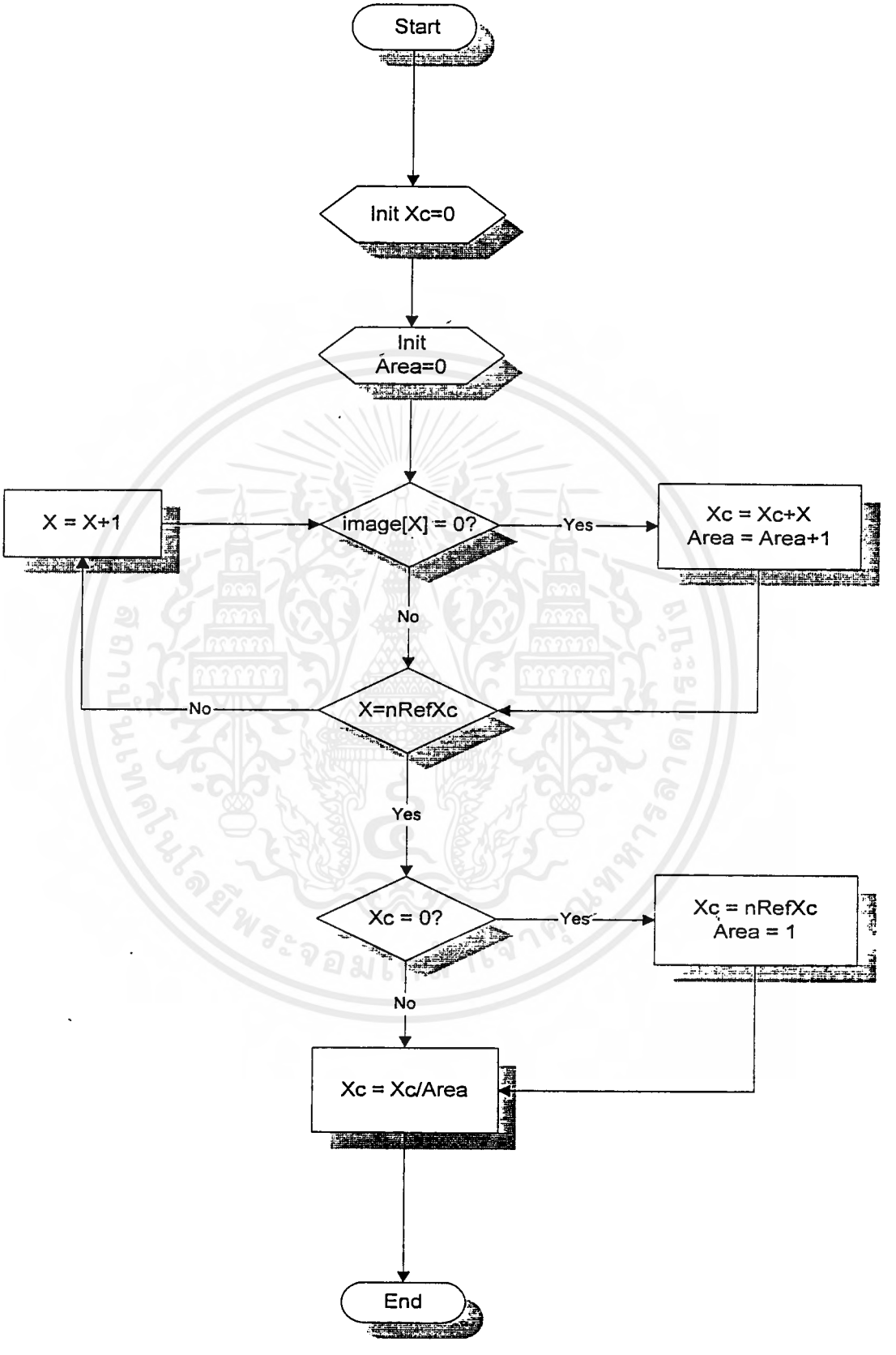
ดังนั้นสัญญาณพัลส์จะมีความถี่ประมาณ 134 Hz ตามต้องการ



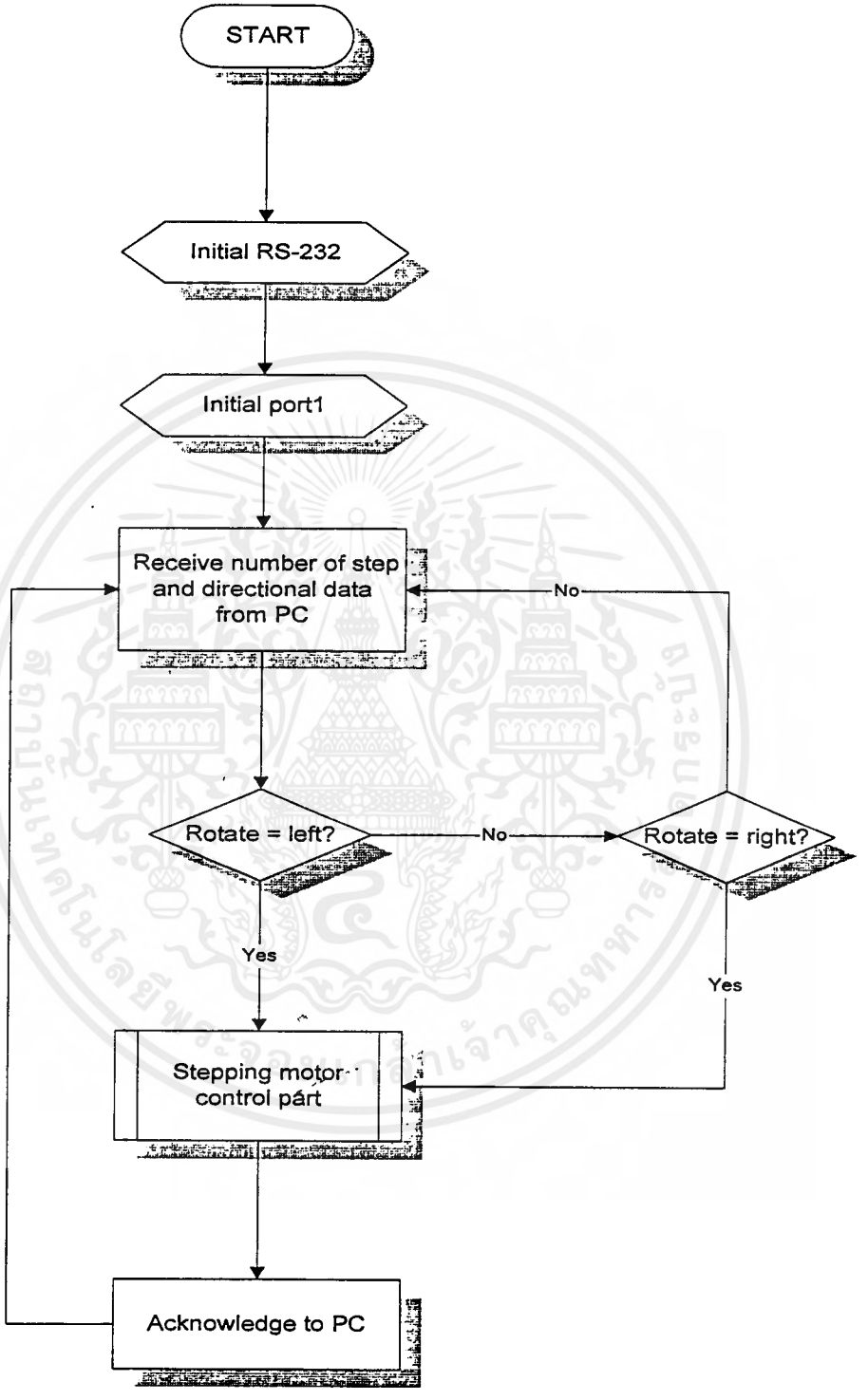
รูปที่ 3.1 แสดงโฟลว์ชาร์ทการทำงานของกระบวนการติดตามวัตถุ



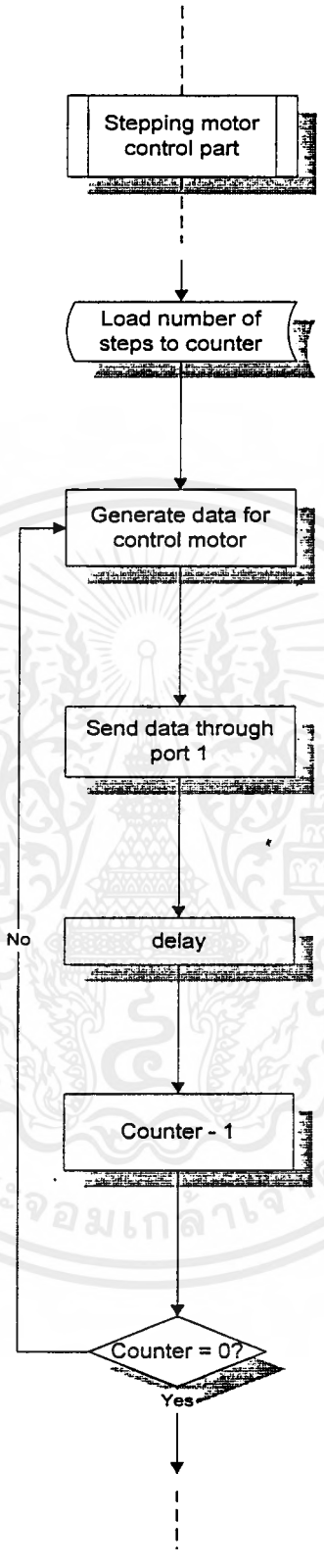
รูปที่ 3.2 แสดงถึงโฟล์วชาร์ทการทำงานของกระบวนการบันทึกภาพวีดีโอ



รูปที่ 3.3 แสดงถึงโฟล์วชาร์ทการทำงานของกระบวนการหาตำแหน่งกึ่งกลางของวัตถุ



รูปที่ 3.4 แสดงถึงโฟล์วชาร์ทการทำงานของกระบวนการควบคุมการหมุนของสเต็ปปิ้งมอเตอร์



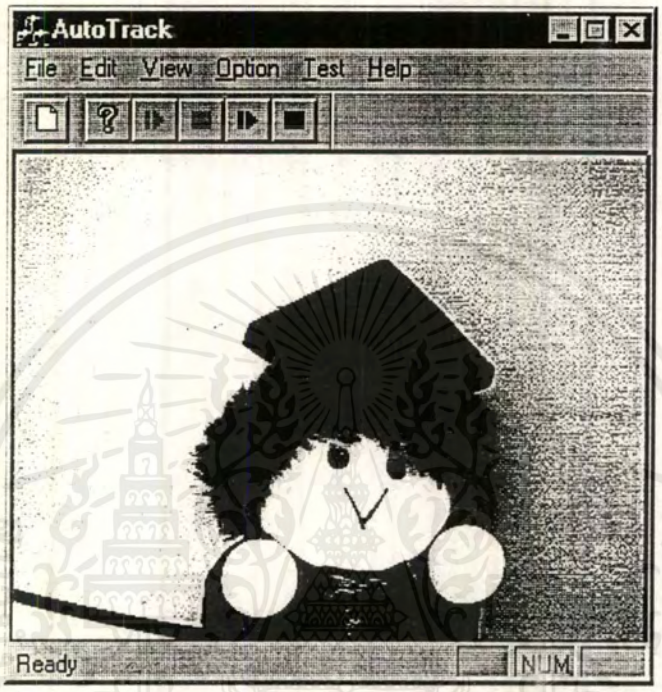
รูปที่ 3.5 แสดงถึงโฟล์วชาร์ทการทำงานของกระบวนการสร้างพัลส์เพื่อควบคุมการหมุนของสเต็ปปิ้งมอเตอร์

บทที่ 4

การทดลองและผลการทดลอง

4.1 แสดงหน้าจอโปรแกรมและเมนูต่างๆ

4.1.1 เมื่อเข้าสู่โปรแกรมเราจะเห็นลักษณะของโปรแกรมดังนี้



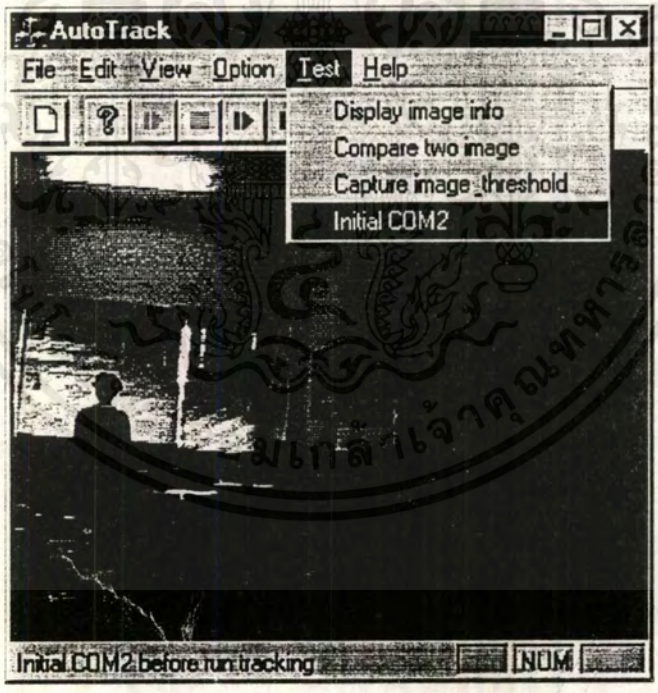
รูปที่ 4.1 แสดงหน้าจอโปรแกรม

4.1.2 ในโปรแกรมจะมีฟังก์ชันการทำงานดังต่อไปนี้

- ฟังก์ชันการติดตามวัตถุอัตโนมัติ
- ฟังก์ชันการบันทึกภาพ
- ฟังก์ชันการถ่ายภาพนิ่งแบบขาว-ดำ
- ฟังก์ชันทดสอบการทำงานของ RS 232
- ฟังก์ชันการทดสอบการเปรียบเทียบภาพสองภาพ

ซึ่งตำแหน่งของฟังก์ชันแสดงดังภาพที่ 4.2 และภาพที่ 4.3

รูปที่ 4.2 แสดงฟังก์ชันในเมนู Option



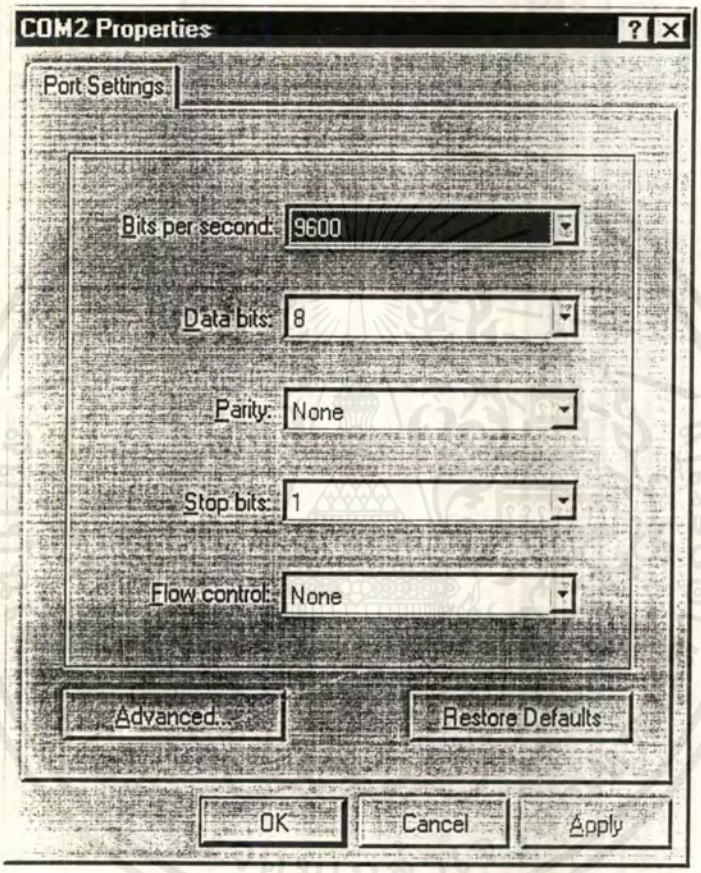
รูปที่ 4.3 แสดงฟังก์ชันในเมนู Test

4.2 การใช้งานโปรแกรมเฉพาะบางฟังก์ชันที่ต้องมีการกำหนดพารามิเตอร์

4.2.1 การใช้งานฟังก์ชันการติดตามวัตถุอัตโนมัติ

1. เมื่อเข้าสู่การทำงานของโปรแกรมหลัก ในการที่จะเริ่มการทำงานในฟังก์ชันนี้นั้นอันดับแรกที่ต้องทำคือการเปิดพอร์ตอนุกรมเสียก่อน โดยการเลือกคำสั่ง "Initial COM2" ที่อยู่ในเมนู Test

2. ทำการกำหนดค่าพารามิเตอร์ในการติดต่อสื่อสารดังเช่นค่า อัตราความเร็วในการรับ-ส่งข้อมูล, การตรวจเช็คความผิดพลาดของข้องมูล, การควบคุม Flow control ซึ่งในโครงงานนี้จะต้องมีการกำหนดค่าต่างๆ ดังแสดงดังรูปข้างล่าง



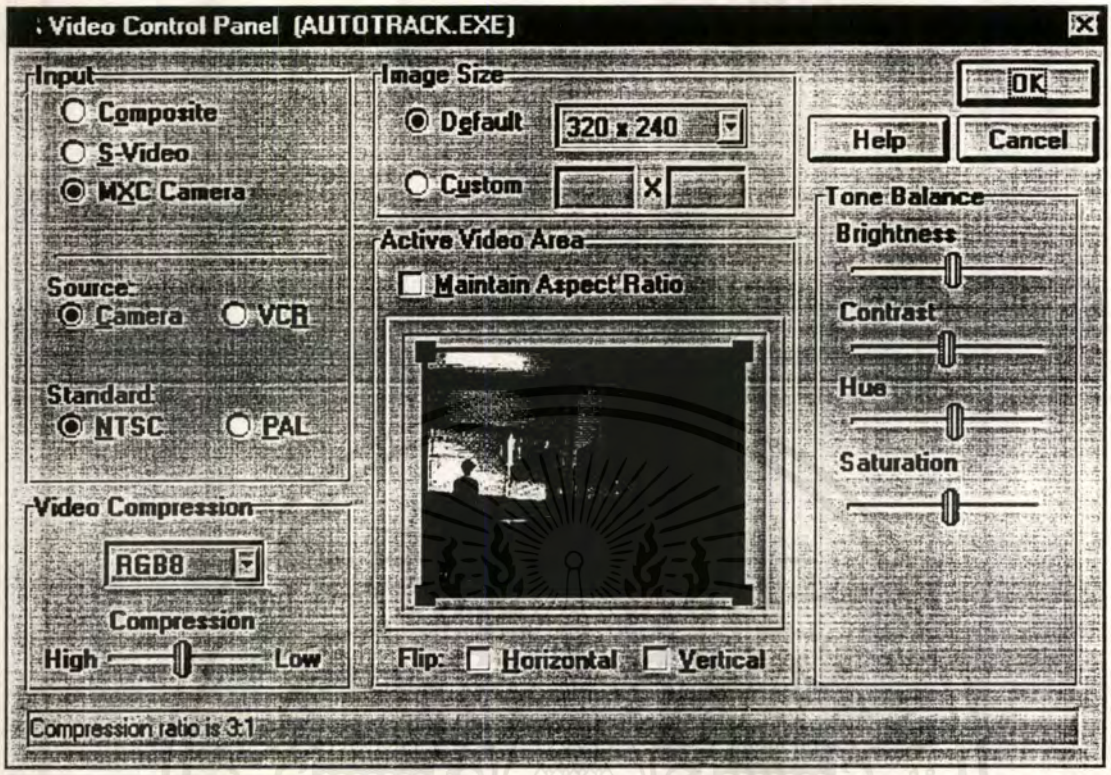
รูปที่ 4.4 แสดง dialog box สำหรับการกำหนด parameters ของพอร์ตอนุกรม

3. เริ่มต้นการทำงานโดยการคลิกปุ่ม "Strat tracking" ที่อยู่ใน Tool bar

4. เมื่อต้องการออกจากการทำงานให้คลิกปุ่ม "Stop tracking" ที่อยู่ใน Tool bar

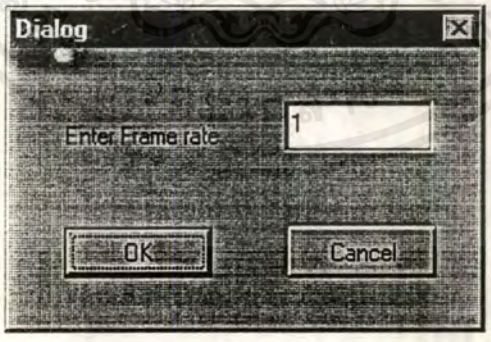
4.2.2 การใช้งานฟังก์ชันการบันทึกภาพวีดีโอ

1. ทำการเลือกขนาดภาพวีดีโอที่ต้องการ โดยการเลือกเมนู" Set video format" ที่อยู่ในเมนู Option



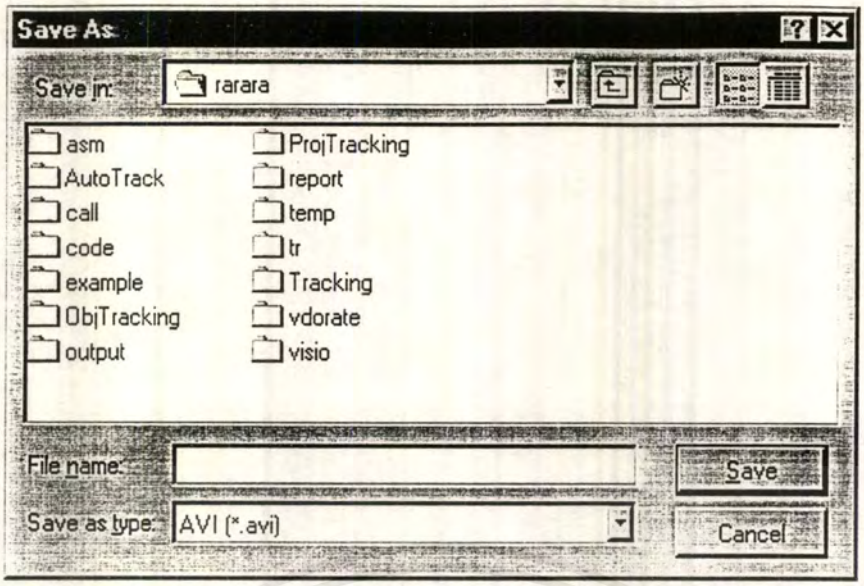
รูปที่ 4.5 แสดง dialog สำหรับเซ็ท video format

2. กำหนดอัตราการบันทึกภาพลงดังรูป ซึ่งใส่ค่าได้ในช่วง 1-30 เฟรมต่อวินาที



รูปที่ 4.6 แสดง dialog box สำหรับกำหนดค่า frame rate

3. กำหนดชื่อไฟล์ตามต้องการ (ไฟล์ข้อมูลจะมีส่วนขยายเป็น "avi")



รูปที่ 4.7 แสดง file save dialog

4. เมื่อต้องการสิ้นสุดการบันทึกวีดีโอให้ทำการคลิกปุ่ม "Stop capture" ที่ Tool bar หรือกด "ESC" ที่คีย์บอร์ด

4.3 ผลการทดลอง

4.3.1 การเปรียบเทียบเฟรมสองเฟรมที่เหมือนกัน

เมื่อวัตถุในภาพไม่มีการเปลี่ยนแปลงใดๆ จะทำให้ภาพที่ได้จากการเปรียบเทียบนั้นปรากฏเป็นภาพที่ว่างเปล่าดังรูป

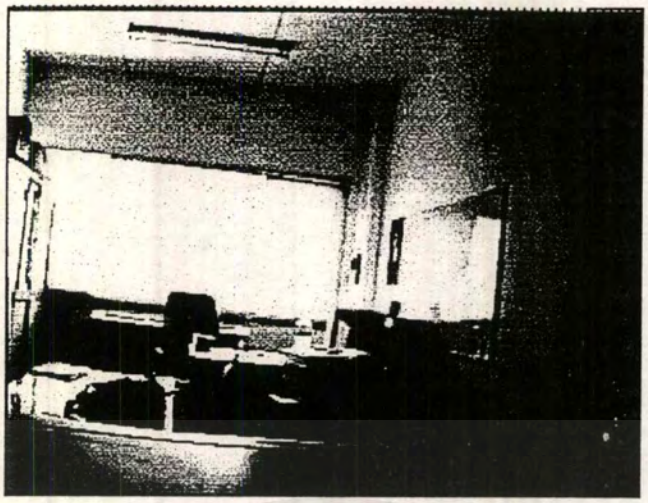
รูปที่ 4.8 แสดงภาพต้นแบบ

รูปที่ 4.9 แสดงภาพที่จะนำมาเปรียบเทียบ

รูปที่ 4.10 แสดงภาพที่ได้จากการเปรียบเทียบภาพที่เหมือนกัน

4.3.2 การเปรียบเทียบเฟรมสองเฟรมที่ต่างกัน

เมื่อวัตถุในภาพมีการเปลี่ยนแปลงใดๆ เกิดขึ้น จะทำให้ภาพที่ได้จากการเปรียบเทียบนั้นปรากฏเป็นภาพที่แสดงถึงการเปลี่ยนแปลงของภาพต้นแบบดังแสดงในรูป



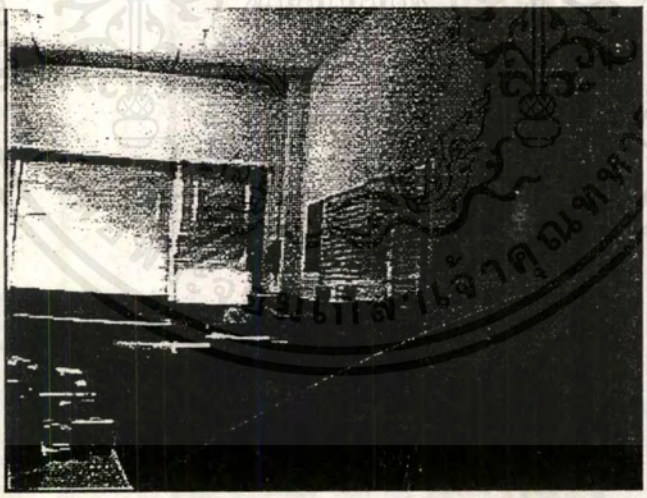
รูปที่ 4.11 แสดงภาพต้นแบบ

รูปที่ 4.12 แสดงภาพที่จะนำมาเปรียบเทียบ

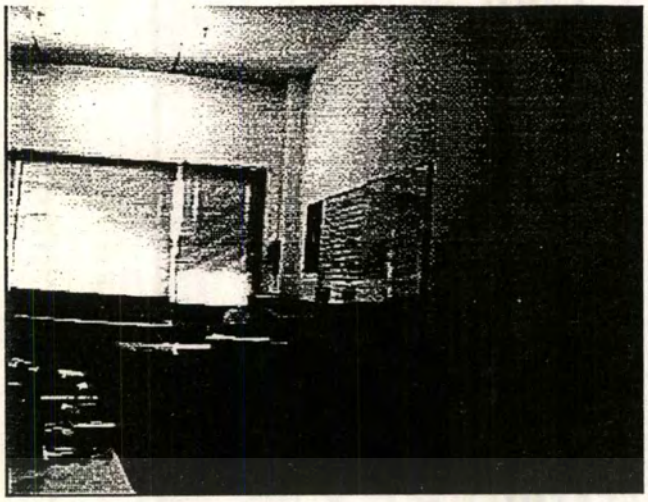
รูปที่ 4.13 แสดงภาพที่ได้จากการเปรียบเทียบ

4.3.3 แสดงการผิดพลาดของการเปรียบเทียบภาพ

ในกรณีที่แสงสว่างในบริเวณที่ทำการจับภาพนั้นมีค่าไม่สม่ำเสมออาจจะทำให้เกิดการผิดพลาดของการตรวจสอบการเปลี่ยนแปลงของวัตถุ ซึ่งอาจได้ค่าตำแหน่งของวัตถุไม่ตรงกับความเป็นจริงดังรูปที่ 4.17 ผลการเปรียบเทียบภาพจะมีจุดดำๆ เกิดขึ้นเนื่องจากความเข้มของแสงไม่คงที่



รูปที่ 4.14 แสดงภาพต้นแบบ

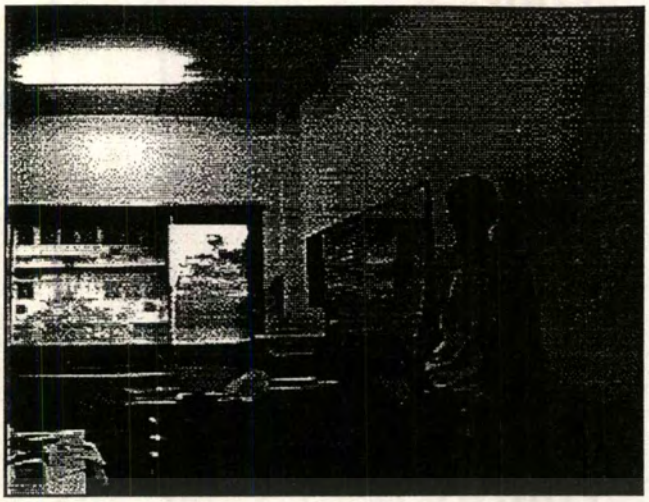


รูปที่ 4.15 แสดงภาพที่จะนำมาเปรียบเทียบ

รูปที่ 4.16 แสดงภาพที่ได้จากการเปรียบเทียบในกรณีที่แสงไม่คงที่

4.3.4 แสดงตัวอย่างการทำ Threshold ของภาพ

ในการทำงานของโปรแกรมนั้นเพื่อที่จะลดความผิดพลาดในการเปรียบเทียบภาพที่เกิดจากแสงเราจำเป็นต้องมีการทำ Threshold ของภาพเสียก่อน แต่วิธีนี้ก็ยังไม่สามารถลดความผิดพลาดได้ทั้งหมด รูปข้างล่างแสดงถึงการทำ Threshold ของภาพ



รูปที่ 4.17 แสดงถึงภาพต้นแบบ

รูปที่ 4.18 แสดงถึงผลของการ Threshold

บทที่ 5
บทสรุปและวิจารณ์

5.1 บทสรุป

จากการทดลองที่ผ่านมามีข้อผิดพลาดที่เด่นชัดก็คือข้อผิดพลาดที่เกิดจากการเปรียบเทียบภาพโดยวิธีหาความแตกต่างของภาพ (การนำภาพสองภาพมาทำการลบกันเพื่อหาความแตกต่างของภาพ) จากผลการทดลองในบทที่4 หัวข้อที่ 4.3.3 เราจะพบว่าผลการเปรียบเทียบภาพและหาตำแหน่งของวัตถุได้ผลเกือบสมบูรณ์แบบถ้าแสงสว่างภายในบริเวณที่ทำการเปรียบเทียบภาพมีความคงที่แต่ถ้าแสงสว่างภายในบริเวณนั้นไม่มีความคงที่ เช่นอาจจะเกิดจากการกระพริบของหลอดไฟภายในห้อง จะทำให้ผลของการเปรียบเทียบภาพที่ผิดไปซึ่งมีผลทำให้หาตำแหน่งของวัตถุผิดพลาดไป และอาจทำให้กล้องเกิดการหมุนทั้งๆ ที่วัตถุไม่ได้มีการเคลื่อนที่ใดๆ เลย

5.2 แนวทางพัฒนา

5.2.1 การข้อผิดพลาดที่เกิดจากแสงนั้นเราสามารถทำการแก้ไขได้โดยทำการเปลี่ยนวิธีการหาตำแหน่งของวัตถุได้โดยการใช้ Motion vector ของภาพ ซึ่งวิธีนี้จะขจัดปัญหาของแสงไปได้และสามารถได้ทั้งขนาดและทิศทางการเคลื่อนไหวของวัตถุ แต่วิธีนี้เป็นเทคนิคขั้นสูงฉะนั้นในโครงงานนี้จึงยังไม่นำมาใช้

5.2.2 เนื่องจากขนาดไฟล์แบบ avi มีความใหญ่มาก ซึ่งไม่เหมาะแก่การทำการบันทึกไว้ดังนั้นเราสามารถลดขนาดไฟล์นี้ได้โดยทำการบีบอัดไฟล์โดยวิธีการใช้การเข้ารหัสแบบ H261 หรือ MPEG

ภาคผนวก

ตารางที่ 1ผ แสดงหน้าที่ของขาสัญญาณแต่ละขาของ connector แบบ DB-25 ในมาตรฐาน RS-232-C

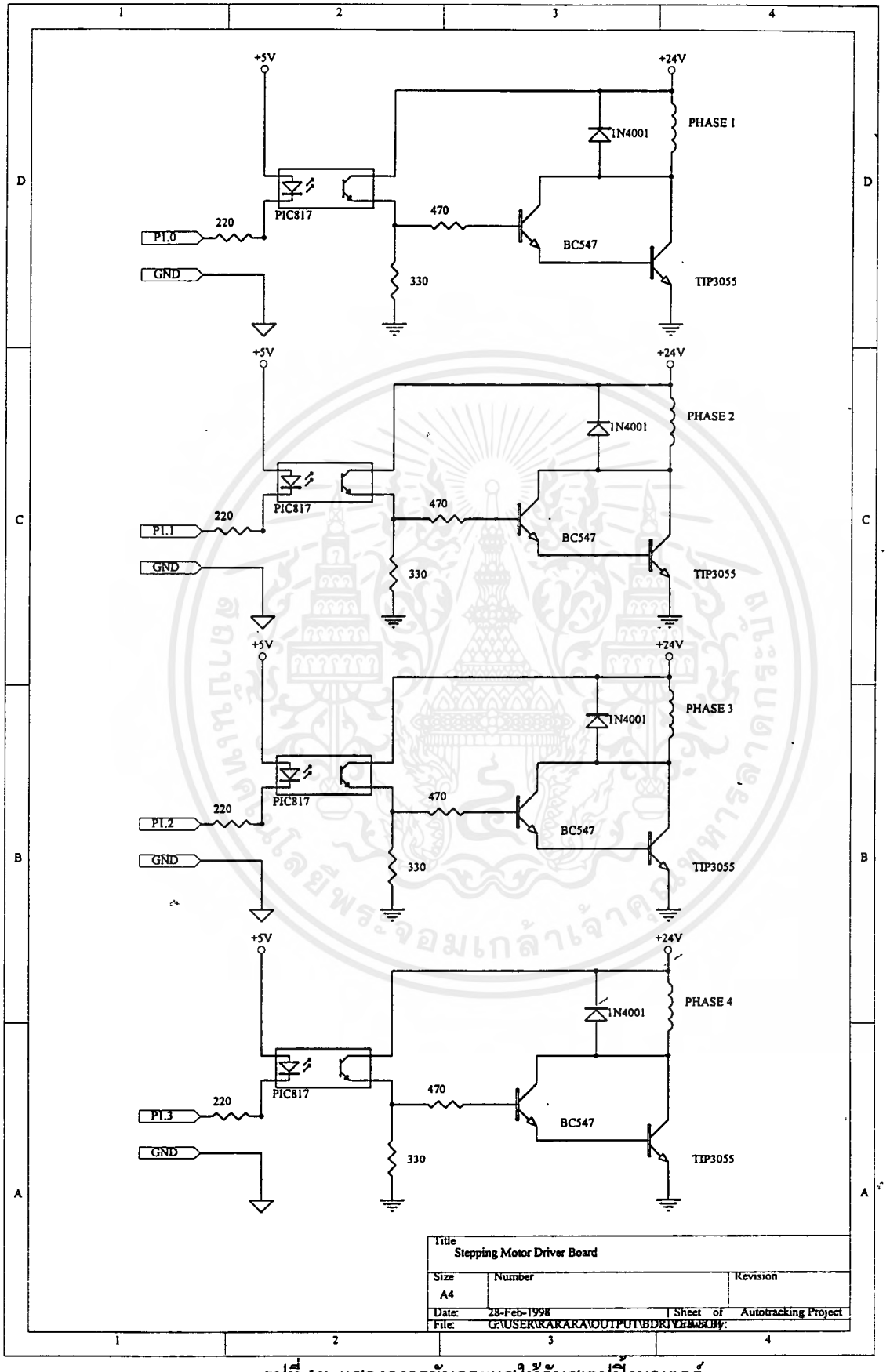
Interchange circuit	Connector pin assignment	Description	Gnd	Data		Control		Timing	
				From DCE	to DCE	From DCE	to DCE	From DCE	to DCE
AA	1	Protective Ground	X						
AB	7	Signal Ground/Common Return	X						
BA	2	Transmitted Data			X				
BB	3	Received Data		X					
	4	Request to Send					X		
	5	Clear to Send				X			
	6	Data Set Ready				X			
	20	Data Terminal Ready					X		
	22	Ring Indicator				X			
	8	Received Line Signal Detector				X			
	21	Signal Quality Detector				X			
	23	Data Signal Rate Selector (DTE)					X		
	23	Data Signal Rate Selector (DCE)				X			
DA	24	Transmitter Signal Element Timing (DTE)							X
DB	15	Transmitter Signal Element Timing (DCE)						X	
DD	17	Receiver Signal Element Timing (DCE)						X	
SBA	14	Secondary Transmitted Data			X				
SBB	16	Secondary Received Data		X					
SCA	19	Secondary Request to Send					X		
SCB	13	Secondary Clear to Send				X			
SCF	12	Secondary Received Line Signal Detector				X			

ตารางที่ 2ผ แสดงรายละเอียดของขาต่าง ๆ พร้อมด้วยสัญญาณกำกับของคอนเนคเตอร์

ขา	เซอร์กิต	ความหมายของเซอร์กิต
1	AA	Protective Ground
2	BA	Transmitted Data
3	BB	Received Data
4	CA	Request to Send
5	CB	Clear to Send
6	CC	Data set Ready
7	AB	Signal Ground
8	CF	Received Line Signal Detector
9/10	-	(Reserved for Data Set Testing)
11	-	Unassigned
12	SCF	Secondary Received Line Signal Dector
13	SCB	Secondary Clear to Send
14	SBA	Secondary Transmitted Data
15	DB	Transmit Signal Element Timing (DCE Source)
16	SBB	Secondary Received Data
17	DD	Receive Signal Element Timing
18	-	Unassigned
19	SCA	Secondary Request to Send
20	CD	Data Terminal Ready
21	CG	Signal Quality Detector
22	CE	Ring Indicator
23	CH/CI	Data Signal Rate Select (DTE/DCE Source)
24	DA	Transmit Signal Element Timing (DTE Source)
25	-	Unassigned

ตารางที่ 3ผ แสดง RS-232-C Standard Configuration

RS-232-C Interchange circuit	Transmit only	Terminal only with RTS	Receive only	Half duplex	Full duplex	Full Duplex with RTS
Spacial						
1 0 Protective Ground	-	-	-	-	-	-
7 x Signal Ground	x	x	x	x	x	x
2 0 Transmitted Data	x	x		x	x	x
3 0 Received Data			x	x	x	x
4 0 Request to Send		x		x		x
5 0 Clear to Send	x	x		x	x	x
6 0 Data Set Ready	x	x	x	x	x	x
20 0 Data Terminal Ready	s	s	s	s	s	s
22 0 Ring Indicator	s	s	s	s	s	s
8 0 Received Line Signal Detector			x	x	x	x



รูปที่ 1ผ แสดงวงจรขับกระแสให้กับสเตปปิ้งมอเตอร์

โปรแกรมภาษา C++ บนวินโดวส์

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#define VC_EXTRALEAN            // Exclude rarely-used stuff from Windows
headers

#include <afxwin.h>          // MFC core and standard components
#include <afxext.h>          // MFC extensions
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                  // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <vfw.h>
#pragma comment(lib,"vfw32.lib")
```

```
// AutoTrack.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "AutoTrack.h"

#include "MainFrm.h"
#include "AutoTrackDoc.h"
#include "AutoTrackView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackApp

BEGIN_MESSAGE_MAP(CAutoTrackApp, CWinApp)
	//{{AFX_MSG_MAP(CAutoTrackApp)
	ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
	// NOTE - the ClassWizard will add and remove mapping macros here.
	//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG_MAP
	// Standard file based document commands
	ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
	ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
	// Standard print setup command
	ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackApp construction

CAutoTrackApp::CAutoTrackApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CAutoTrackApp object

CAutoTrackApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackApp initialization

BOOL CAutoTrackApp::InitInstance()
{
	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a
shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC
statically
#endif

	LoadStdProfileSettings();  // Load standard INI file options (including
MRU)
```

```
	// Register the application's document templates.  Document templates
	//  serve as the connection between documents, frame windows and views.

	CSingleDocTemplate* pDocTemplate;
	pDocTemplate = new CSingleDocTemplate(
	IDR_MAINFRAME,
	RUNTIME_CLASS(CAutoTrackDoc),
	RUNTIME_CLASS(CMainFrame),       // main SDI frame window
	RUNTIME_CLASS(CAutoTrackView));
	AddDocTemplate(pDocTemplate);

	// Parse command line for standard shell commands, DDE, file open
	CCommandLineInfo cmdInfo;
	ParseCommandLine(cmdInfo);

	// Dispatch commands specified on the command line
	if (!ProcessShellCommand(cmdInfo))
		return FALSE;

	return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	// No message handlers
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
	// No message handlers
```

```
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CAutoTrackApp::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackApp commands
```

```
// CaptureTh.h : header file
//
#ifndef _CaptureTh_
#define _CaptureTh_

class CAutoTrackDoc;


/////////////////////////////////////////////////////////////////////////////
// CaptureTh thread

class CaptureTh : public CWinThread
{
      DECLARE_DYNCREATE(CaptureTh)
protected:
      CaptureTh();           // protected constructor used by dynamic creation

// Attributes
public:
   CAutoTrackDoc* m_pOwner;
   HANDLE hEventKill;
   HANDLE hEventDead;


// Operations
public:
   void SetOwner(CAutoTrackDoc* pOwner) {m_pOwner=pOwner;};
   void KillThread();
   void ClearUpThread();

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CaptureTh)
      public:
      virtual BOOL InitInstance();
      virtual int ExitInstance();
      virtual int Run();
      //}}AFX_VIRTUAL

// Implementation
protected:
      virtual ~CaptureTh();

      // Generated message map functions
      //{{AFX_MSG(CaptureTh)
      // NOTE - the ClassWizard will add and remove member functions here.
      //}}AFX_MSG

      DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
#endif _CaptureTh_
```

```
// CaptureTh.cpp : implementation file
//

#include "stdafx.h"
#include "AutoTrack.h"
#include "CaptureTh.h"
#include "AutoTrackDoc.h"
#include "SetFrDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CaptureTh

IMPLEMENT_DYNCREATE(CaptureTh, CWinThread)

CaptureTh::CaptureTh()
{
   hEventKill=CreateEvent(NULL,TRUE,FALSE,NULL);
   hEventDead=CreateEvent(NULL,TRUE,FALSE,NULL);
}

CaptureTh::~CaptureTh()
{
   CloseHandle(hEventKill);
   CloseHandle(hEventDead);
}

BOOL CaptureTh::InitInstance()
{
      // TODO:  perform and per-thread initialization here
      return TRUE;
}

int CaptureTh::ExitInstance()
{
      // TODO:  perform any per-thread cleanup here
      return CWinThread::ExitInstance();
}

BEGIN_MESSAGE_MAP(CaptureTh, CWinThread)
      //{{AFX_MSG_MAP(CaptureTh)
      // NOTE - the ClassWizard will add and remove mapping macros here.
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CaptureTh message handlers
int CaptureTh::Run()
{
   if (m_pOwner==NULL)
      return (-1);

      AfxMessageBox("HI");

   CString PathName;
      CAPTUREPARMS CaptureParms;
   SetFrDlg SetrateDlg;


   int response;
   float FramesPerSec ;
```

```
    CFileDialog* FSaveDlg;

    FSaveDlg = new
    CFileDialog(FALSE,"AVI","",OFN_OVERWRITEPROMPT|OFN_HIDEREADONLY,"AVI
    (*.avi)|*.avi|All Files(*.*)|*.*||");

    if ( SetrateDlg.DoModal()==IDOK)
    {

        FramesPerSec=(float)SetrateDlg.m_FrameRate;
        response=FSaveDlg->DoModal();
        if (response==IDOK)
        {
            PathName = FSaveDlg->GetPathName();
            capFileSetCaptureFile(m_pOwner->hWndC,(LPCSTR)PathName);
            capCaptureGetSetup(m_pOwner->hWndC, &CaptureParms,
sizeof(CAPTUREPARMS));
            CaptureParms.fAbortLeftMouse=FALSE;
            CaptureParms.dwRequestMicroSecPerFrame = (DWORD) (1.0e6 /
FramesPerSec);
            capCaptureSetSetup(m_pOwner->hWndC, &CaptureParms, sizeof
(CAPTUREPARMS));

            capCaptureSequence(m_pOwner->hWndC);

            delete FSaveDlg;

        }
    }

    else
    {
       delete FSaveDlg;
       ClearUpThread();
       //AfxEndThread(0);
       return 0;
    }
    if(WaitForSingleObject(hEventKill,INFINITE)==WAIT_OBJECT_0)
    {
       //VERIFY(SetEvent(hEventDead));
       AfxMessageBox("End capture thread");
       ClearUpThread();
       AfxEndThread(0);
       return 0;
    }
       else return 1;
}


void CaptureTh::KillThread()
{
   VERIFY(SetEvent(hEventKill));
   SetThreadPriority(THREAD_PRIORITY_ABOVE_NORMAL);

}

void CaptureTh::ClearUpThread()
{
   m_pOwner->m_pCaptureth=NULL;
}
```

```
// AutoTrack.h : main header file for the AUTOTRACK application
//

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackApp:
// See AutoTrack.cpp for the implementation of this class
//

class CAutoTrackApp : public CWinApp
{
public:
	CAutoTrackApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAutoTrackApp)
public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CAutoTrackApp)
	afx_msg void OnAppAbout();
	// NOTE - the ClassWizard will add and remove member functions here.
	//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////
```

```
// AutoTrack.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "AutoTrack.h"

#include "MainFrm.h"
#include "AutoTrackDoc.h"
#include "AutoTrackView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackApp

BEGIN_MESSAGE_MAP(CAutoTrackApp, CWinApp)
	//{{AFX_MSG_MAP(CAutoTrackApp)
	ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
	// NOTE - the ClassWizard will add and remove mapping macros here.
	//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG_MAP
	// Standard file based document commands
	ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
	ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
	// Standard print setup command
	ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackApp construction

CAutoTrackApp::CAutoTrackApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CAutoTrackApp object

CAutoTrackApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackApp initialization

BOOL CAutoTrackApp::InitInstance()
{
	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a
shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC
statically
#endif

	LoadStdProfileSettings();  // Load standard INI file options (including
MRU)
```

```
	// Register the application's document templates.  Document templates
	//  serve as the connection between documents, frame windows and views.

	CSingleDocTemplate* pDocTemplate;
	pDocTemplate = new CSingleDocTemplate(
		IDR_MAINFRAME,
		RUNTIME_CLASS(CAutoTrackDoc),
		RUNTIME_CLASS(CMainFrame),       // main SDI frame window
		RUNTIME_CLASS(CAutoTrackView));
	AddDocTemplate(pDocTemplate);

	// Parse command line for standard shell commands, DDE, file open
	CCommandLineInfo cmdInfo;
	ParseCommandLine(cmdInfo);

	// Dispatch commands specified on the command line
	if (!ProcessShellCommand(cmdInfo))
		return FALSE;

	return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
		// No message handlers
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
	// No message handlers
```

```
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CAutoTrackApp::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackApp commands
```

```
// AutoTrackView.h : interface of the CAutoTrackView class
//
/////////////////////////////////////////////////////////////////////////////

class CAutoTrackView : public CView
{
protected: // create from serialization only
      CAutoTrackView();
      DECLARE_DYNCREATE(CAutoTrackView)

// Attributes
public:
      CAutoTrackDoc* GetDocument();

// Operations
public:

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CAutoTrackView)
public:
      virtual void OnDraw(CDC* pDC);  // overridden to draw this view
      virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
      virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
      virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
      virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
      //}}AFX_VIRTUAL

// Implementation
public:
      virtual ~CAutoTrackView();
#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
      //{{AFX_MSG(CAutoTrackView)
      // NOTE - the ClassWizard will add and remove member functions here.
      //    DO NOT EDIT what you see in these blocks of generated code !
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in AutoTrackView.cpp
inline CAutoTrackDoc* CAutoTrackView::GetDocument()
   { return (CAutoTrackDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////
```

```
// AutoTrackView.cpp : implementation of the CAutoTrackView class
//

#include "stdafx.h"
#include "AutoTrack.h"

#include "AutoTrackDoc.h"
#include "AutoTrackView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackView

IMPLEMENT_DYNCREATE(CAutoTrackView, CView)

BEGIN_MESSAGE_MAP(CAutoTrackView, CView)
	//{{AFX_MSG_MAP(CAutoTrackView)
	// NOTE - the ClassWizard will add and remove mapping macros here.
	//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG_MAP
	// Standard printing commands
	ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackView construction/destruction

CAutoTrackView::CAutoTrackView()
{
	// TODO: add construction code here

}

CAutoTrackView::~CAutoTrackView()
{
}

BOOL CAutoTrackView::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackView drawing

void CAutoTrackView::OnDraw(CDC* pDC)
{
	CAutoTrackDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);

	// TODO: add draw code for native data here
}

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackView printing
```

```
BOOL CAutoTrackView::OnPreparePrinting(CPrintInfo* pInfo)
{
	// default preparation
	return DoPreparePrinting(pInfo);
}

void CAutoTrackView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add extra initialization before printing
}

void CAutoTrackView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add cleanup after printing
}

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackView diagnostics

#ifdef _DEBUG
void CAutoTrackView::AssertValid() const
{
	CView::AssertValid();
}

void CAutoTrackView::Dump(CDumpContext& dc) const
{
	CView::Dump(dc);
}

CAutoTrackDoc* CAutoTrackView::GetDocument() // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CAutoTrackDoc)));
	return (CAutoTrackDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackView message handlers
```

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
      CMainFrame();
      DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CMainFrame)
      virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
      //}}AFX_VIRTUAL

// Implementation
public:
      virtual ~CMainFrame();
#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
      CStatusBar  m_wndStatusBar;
      CToolBar    m_wndToolBar;

// Generated message map functions
protected:
      //{{AFX_MSG(CMainFrame)
      afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
      // NOTE - the ClassWizard will add and remove member functions here.
      // DO NOT EDIT what you see in these blocks of generated code!
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "AutoTrack.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
      //{{AFX_MSG_MAP(CMainFrame)
            // NOTE - the ClassWizard will add and remove mapping macros here.
            //    DO NOT EDIT what you see in these blocks of generated code !
      ON_WM_CREATE()
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
      ID_SEPARATOR,           // status line indicator
      ID_INDICATOR_CAPS,
      ID_INDICATOR_NUM,
      ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
      // TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
      if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
            return -1;

      if (!m_wndToolBar.Create(this) ||
            !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
      {
            TRACE0("Failed to create toolbar\n");
            return -1;      // fail to create
      }

      if (!m_wndStatusBar.Create(this) ||
            !m_wndStatusBar.SetIndicators(indicators,
              sizeof(indicators)/sizeof(UINT)))
      {
            TRACE0("Failed to create status bar\n");
            return -1;      // fail to create
```

```
    }

    // TODO: Remove this if you don't want tool tips or a resizeable toolbar
    m_wndToolBar.SetBarStyle(m_wndToolBar.GetBarStyle() |
          CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC);

    // TODO: Delete these three lines if you don't want the toolbar to
    //  be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs
  cs.style=WS_MINIMIZEBOX|WS_SYSMENU|WS_CAPTION|WS_BORDER;

    return CFrameWnd::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
```

```
// Input.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// Input dialog

class Input : public CDialog
{
// Construction
public:
	Input(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(Input)
	enum { IDD = IDD_DIALOG2 };
	float m_size;
	float m_distance;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(Input)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(Input)
	// NOTE: the ClassWizard will add member functions here
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
// Input.cpp : implementation file
//

#include "stdafx.h"
#include "AutoTrack.h"
#include "Input.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// Input dialog


Input::Input(CWnd* pParent /*=NULL*/)
	: CDialog(Input::IDD, pParent)
{
	//{{AFX_DATA_INIT(Input)
	m_size = 320.0f;
	m_distance = 320.0f;
	//}}AFX_DATA_INIT
}


void Input::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(Input)
	DDX_Text(pDX, IDC_SCRDIST, m_size);
	DDV_MinMaxFloat(pDX, m_size, 0.f, 500.f);
	DDX_Text(pDX, IDC_SCRSIZE, m_distance);
	DDV_MinMaxFloat(pDX, m_distance, 0.f, 500.f);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(Input, CDialog)
	//{{AFX_MSG_MAP(Input)
	// NOTE: the ClassWizard will add message map macros here
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// Input message handlers
```

```
// SetFrDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// SetFrDlg dialog

class SetFrDlg : public CDialog
{
// Construction
public:
	SetFrDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
	//{{AFX_DATA(SetFrDlg)
	enum { IDD = IDD_DIALOG1 };
	UINT  m_FrameRate;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(SetFrDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(SetFrDlg)
	// NOTE: the ClassWizard will add member functions here
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

```
// SetFrDlg.cpp : implementation file
//

#include "stdafx.h"
#include "AutoTrack.h"
#include "SetFrDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// SetFrDlg dialog


SetFrDlg::SetFrDlg(CWnd* pParent /*=NULL*/)
	: CDialog(SetFrDlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(SetFrDlg)
	m_FrameRate = 1;
	//}}AFX_DATA_INIT
}


void SetFrDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(SetFrDlg)
	DDX_Text(pDX, IDC_FRVAL, m_FrameRate);
	DDV_MinMaxUInt(pDX, m_FrameRate, 1, 30);
	//}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(SetFrDlg, CDialog)
	//{{AFX_MSG_MAP(SetFrDlg)
	// NOTE: the ClassWizard will add message map macros here
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// SetFrDlg message handlers
```

```
// TrackingTh.h : header file
//
#ifndef _TrackingTh_
#define _TrackingTh_

class CAutoTrackDoc;
/////////////////////////////////////////////////////////////////////////////
// TrackingTh thread

class TrackingTh : public CWinThread
{
      DECLARE_DYNCREATE(TrackingTh)
protected:
      TrackingTh();           // protected constructor used by dynamic
creation

// Attributes
public:
   CAutoTrackDoc* m_pOwner;
   BOOL EXIT;
// Operations
public:
   void SetOwner(CAutoTrackDoc* pOwner) {m_pOwner=pOwner;};
   void KillThread();
   void ClearUpThread();

// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(TrackingTh)
      public:
      virtual BOOL InitInstance();
      virtual int ExitInstance();
      virtual int Run();
   //}}AFX_VIRTUAL

// Implementation
protected:
      virtual ~TrackingTh();

      // Generated message map functions
      //{{AFX_MSG(TrackingTh)
      // NOTE - the ClassWizard will add and remove member functions here.
      //}}AFX_MSG

      DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////
#endif _TrackingTh_
```

```
// TrackingTh.cpp : implementation file
//

#include "stdafx.h"
#include "AutoTrack.h"
#include "TrackingTh.h"
#include "AutoTrackDoc.h"
#include "Input.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define LEFT 0x4b
#define RIGHT 0x4d
HANDLE hEventKill;
HANDLE hEventDead;

/////////////////////////////////////////////////////////////////////////////
// TrackingTh

IMPLEMENT_DYNCREATE(TrackingTh, CWinThread)

TrackingTh::TrackingTh()
{
   hEventKill=CreateEvent(NULL,TRUE,FALSE,NULL);
   hEventDead=CreateEvent(NULL,TRUE,FALSE,NULL);
}

TrackingTh::~TrackingTh()
{
   CloseHandle(hEventKill);
   CloseHandle(hEventDead);
}

BOOL TrackingTh::InitInstance()
{
      // TODO:  perform and per-thread initialization here
      return TRUE;
}

int TrackingTh::ExitInstance()
{
      // TODO:  perform any per-thread cleanup here
      return CWinThread::ExitInstance();
}

BEGIN_MESSAGE_MAP(TrackingTh, CWinThread)
      //{{AFX_MSG_MAP(TrackingTh)
      // NOTE - the ClassWizard will add and remove mapping macros here.
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// TrackingTh message handlers
int TrackingTh::Run()
{
   if (m_pOwner==NULL)
      return (-1);

   UINT nWidth;
   int nXc;
   int nDeltaXc;
   //char nInData;
```

```
Input  mInput;
float temp1;
float temp2,nStep;
char nOutData;
char indata;
DWORD dwLength;


if (mInput.DoModal()==IDOK)
{
   AfxMessageBox("Hi");

   nWidth=m_pOwner->lpbInfoHeader->biWidth;

   m_pOwner->ReadRefFrame();
   m_pOwner->ToGrayScale(m_pOwner->pbFrame1);
   do
   {
         do
         {
               m_pOwner->ReadSecondFrame();
               m_pOwner->ToGrayScale(m_pOwner->pbFrame2);
               m_pOwner->CompareImage();
               nXc=m_pOwner->FindCentroidOfImage();
               if(WaitForSingleObject(hEventKill,0)==WAIT_OBJECT_0)
               {
                     ClearUpThread();
                     AfxMessageBox("End tracking thread");
                     return(0);
               }
         }
         while  (nXc==m_pOwner->nRefXc);
         if (nXc<m_pOwner->nRefXc)
         {
               nDeltaXc=(int)(m_pOwner->nRefXc-nXc);
               temp1=(DWORD)(mInput.m_distance*nWidth);
               temp2=(double)( (152.789*mInput.m_size)/temp1 );

               nStep=(LONG)(nDeltaXc*temp2);
               nOutData=(unsigned char)(nStep);
               m_pOwner->WriteCommBlock(LEFT);
               m_pOwner->WriteCommBlock(&nOutData);
         }
         if (nXc>m_pOwner->nRefXc)
         {
               nDeltaXc=(int)(nXc-m_pOwner->nRefXc);
               temp1=(DWORD)(mInput.m_distance*nWidth);
               temp2=(double)( (152.789*mInput.m_size)/temp1 );

               nStep=(LONG)(nDeltaXc*temp2);
               nOutData=(unsigned char)(nStep);
               m_pOwner->WriteCommBlock(RIGHT);
               m_pOwner->WriteCommBlock(&nOutData);
         }

         indata=0;

         urgeComm(m_pOwner->hCOM2,PURGE_RXCLEAR);

         do
         {
               ReadFile( m_pOwner->hCOM2,&indata,1,&dwLength,&(m_pOwner-
   >o));
               if(WaitForSingleObject(hEventKill,0)==WAIT_OBJECT_0)
               {
                     ClearUpThread();
```

```
                AfxMessageBox("End tracking thread");
                return(0);
            }
        }
        while(indata==0);


        m_pOwner->ReadRefFrame();
        m_pOwner->ToGrayScale(m_pOwner->pbFrame1);

    }
    while (WaitForSingleObject(hEventKill,0)!=WAIT_OBJECT_0);


  }
  ClearUpThread();
  AfxMessageBox("End tracking thread");


  return 0;


}// end function run and return 0



void TrackingTh::KillThread()
{
  VERIFY(SetEvent(hEventKill));
  SetThreadPriority(THREAD_PRIORITY_ABOVE_NORMAL);
  //WaitForSingleObject(hEventDead,INFINITE);
}

void TrackingTh::ClearUpThread()
{
  m_pOwner->m_pTrackingTh=NULL;
}
```

```
// AutoTrackDoc.h : interface of the CAutoTrackDoc class
//
/////////////////////////////////////////////////////////////////////////////
//#ifndef _CAutoTrackDoc_
//#define _CAutoTrackDoc_

class TrackingTh;
class CaptureTh;
/////////////////////////////////////////////////////////////////////////////
class CAutoTrackDoc : public CDocument
{
protected: // create from serialization only
      CAutoTrackDoc();
      DECLARE_DYNCREATE(CAutoTrackDoc)

// Attributes
public:
      HWND hWndC;
      CaptureTh* m_pCaptureth;
      TrackingTh* m_pTrackingTh;
      BITMAPINFO* bitMapInfo;
      BYTE* pbVideoFormat;
      BYTE* pbFrame1;
      BYTE* pbFrame2;
      BYTE* pbResultFrame;
      RGBQUAD* pRGB;
      int nXc;
      int nRefXc;
      //WORD nBitPerPixel;
      LPBITMAPINFOHEADER lpbInfoHeader;
      HANDLE hCOM2;
      COMMCONFIG lpCC;
      OVERLAPPED o;
      char indata;

// Operations
public:
      void RGB2GrayScale(BYTE* pbGrayFrame);
      void ReadRefFrame();
      void ReadSecondFrame();
      void ToGrayScale(BYTE* pGrayFrmame);
      void CompareImage();
      void Threshold(BYTE* pBuffFrmame);
      LONG FindCentroidOfImage();
      BYTE FindMaxPixelValue(BYTE* pBuffFrmame);
      BOOL InitRS232();
      BOOL WriteCommBlock(LPSTR lpByte);
      int  ReadCommBlock(LPSTR lpszBlock, int nMaxLength );
      BOOL CloseConnection();
// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CAutoTrackDoc)
public:
      virtual BOOL OnNewDocument();
      virtual void Serialize(CArchive& ar);
      //}}AFX_VIRTUAL

// Implementation
public:
      virtual ~CAutoTrackDoc();
#ifdef _DEBUG
      virtual void AssertValid() const;
      virtual void Dump(CDumpContext& dc) const;
#endif

protected:
```

```
    static LRESULT CALLBACK VideoFrameCallback(HWND hWnd,LPVIDEOHDR lpVHdr);

// Generated message map functions
protected:
      //{{AFX_MSG(CAutoTrackDoc)
      afx_msg void OnCapture();
      afx_msg void OnStopcapture();
      afx_msg void OnTrack();
      afx_msg void OnStoptrack();
      afx_msg void OnCheckdisk();
      afx_msg void OnSetformat();
      afx_msg void OnChecktime();
      afx_msg void OnTestCaptureGrayImage();
      afx_msg void OnInfo();
      afx_msg void OnCompare();
      afx_msg void OnCapthres();
      afx_msg void OnInitcom2();
      afx_msg void OnTest232();
      afx_msg void OnCapcolor();
      afx_msg void OnUpdateStopcapture(CCmdUI* pCmdUI);
      afx_msg void OnUpdateStoptrack(CCmdUI* pCmdUI);
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
//#endif _CAutoTrackDoc_
```

```
// AutoTrackDoc.cpp : implementation of the CAutoTrackDoc class
//

#include "stdafx.h"
#include "AutoTrack.h"

#include "TrackingTh.h"
#include "CaptureTh.h"

#include "AutoTrackDoc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

static      BYTE* pbBuffer;
static      LONG nHeight;
static      LONG nWidth;

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackDoc

IMPLEMENT_DYNCREATE(CAutoTrackDoc, CDocument)

BEGIN_MESSAGE_MAP(CAutoTrackDoc, CDocument)
      //{{AFX_MSG_MAP(CAutoTrackDoc)
      ON_COMMAND(ID_CAPTURE, OnCapture)
      ON_COMMAND(ID_STOPCAPTURE, OnStopcapture)
      ON_COMMAND(ID_TRACK, OnTrack)
      ON_COMMAND(ID_STOPTRACK, OnStoptrack)
      ON_COMMAND(IDM_CHECKDISK, OnCheckdisk)
      ON_COMMAND(IDM_SETFORMAT, OnSetformat)
      ON_COMMAND(IDM_CHECKTIME, OnChecktime)
      ON_COMMAND(IDM_TESTCAPTURE, OnTestCaptureGrayImage)
      ON_COMMAND(IDM_INFO, OnInfo)
      ON_COMMAND(IDM_COMPARE, OnCompare)
      ON_COMMAND(IDM_CAPTHRES, OnCapthres)
      ON_COMMAND(IDM_INITCOM2, OnInitcom2)
      ON_COMMAND(IDM_TEST232, OnTest232)
      ON_COMMAND(IDM_CAPCOLOR, OnCapcolor)
      ON_UPDATE_COMMAND_UI(ID_STOPCAPTURE, OnUpdateStopcapture)
      ON_UPDATE_COMMAND_UI(ID_STOPTRACK, OnUpdateStoptrack)
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackDoc construction/destruction

CAutoTrackDoc::CAutoTrackDoc()
{
      // TODO: add one-time construction code here

      m_pCaptureth=NULL;
      m_pTrackingTh=NULL;

}

CAutoTrackDoc::~CAutoTrackDoc()
{
      free(pbFrame1);
      free(pbFrame2);
      free(pbVideoFormat);
      free(pbResultFrame);
      CloseHandle(hCOM2);
```

```
        CloseHandle(o.hEvent);
}

BOOL CAutoTrackDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;

        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)

        CView* pView;
        CWnd*  pFrame;
        POSITION pos;
        BOOL   canConnect;

        pos=GetFirstViewPosition();
        pView=(CView*)GetNextView(pos);
        pFrame=pView->GetParentFrame();
        hWndC =
capCreateCaptureWindow("CAPTURE",WS_CHILD|WS_VISIBLE,10,35,300,300,pFrame-
>GetSafeHwnd(),0);
        canConnect=capDriverConnect(hWndC,1);

        if (canConnect==FALSE)
        {
                AfxMessageBox("Can't connect to driver");
                return FALSE;

        }

        CAPSTATUS CapStatus;

    // get status of capture window
    // calculate initial size of view

        capGetStatus(hWndC,&CapStatus,sizeof(CAPSTATUS));

        ::MoveWindow(hWndC,0,GetSystemMetrics(SM_CYICON),(int)CapStatus.uiImageW
        idth,(int)CapStatus.uiImageHeight,FALSE);
        CRect client,window;

        pFrame->GetClientRect(&client);
        pFrame->GetWindowRect(&window);
        client.right =client.left+window.Width() -client.Width()
        +(int)CapStatus.uiImageWidth;
        client.bottom=client.top +window.Height()-
        client.Height()+(int)CapStatus.uiImageHeight+GetSystemMetrics(SM_CYICON)
        +GetSystemMetrics(SM_CYSIZE);
        pFrame-
>MoveWindow(client.left,client.top,client.right,client.bottom,TRUE);
    pFrame->CenterWindow(NULL);

    // display video image
    // 40=25 fps, 50=20 fps, 67=15 fps

    if (capPreviewRate(hWndC,40))
    {

        capPreview(hWndC,TRUE);

    }

    // Set callback function
    BOOL bSetCallbackStatus;
```

```
  bSetCallbackStatus=capSetCallbackOnFrame(hWndC,VideoFrameCallback);
     if (bSetCallbackStatus==FALSE)
     {
        AfxMessageBox("Error, can't set callback function");
  }

  // Access to videoformat header
     pbVideoFormat=(BYTE*)malloc(capGetVideoFormatSize(hWndC)*sizeof(BYTE));
     if (pbVideoFormat==NULL)
     {
           AfxMessageBox("Can't allocate memory for 'pbVideoFormat' ");
           exit(1);
     }

     lpbInfoHeader=(LPBITMAPINFOHEADER)pbVideoFormat;
     capGetVideoFormat(hWndC,lpbInfoHeader,capGetVideoFormatSize(hWndC));
     pRGB=(RGBQUAD*)(pbVideoFormat+lpbInfoHeader->biSize);
  nHeight=lpbInfoHeader->biHeight;
     nWidth=lpbInfoHeader->biWidth;
     nRefXc=(int)(nWidth/2);

     // Allocate memory blocks
     pbBuffer=(BYTE*)malloc(lpbInfoHeader->biWidth*lpbInfoHeader->biHeight);
     if (pbBuffer==NULL)
     {
           AfxMessageBox("Can't allocate memory for 'pbBuffer' ");
           return FALSE;
     }

     pbFrame1=(BYTE*)malloc(lpbInfoHeader->biWidth*lpbInfoHeader->biHeight);
     if (pbFrame1==NULL)
     {
           AfxMessageBox("Can't allocate memory for 'pbFrame1' ");
           return FALSE;
     }

     pbFrame2=(BYTE*)malloc(lpbInfoHeader->biWidth*lpbInfoHeader->biHeight);
     if (pbFrame2==NULL)
     {
           AfxMessageBox("Can't allocate memory for 'pbFrame2' ");
           return FALSE;
     }

  pbResultFrame=(BYTE*)malloc(lpbInfoHeader->biWidth*lpbInfoHeader-
>biHeight);
     if (pbFrame2==NULL)
     {
           AfxMessageBox("Can't allocate memory for 'pbResultFrame' ");
           return FALSE;

     }



     return TRUE;
}

//////////////////////////////////////////////////////////////////////////
// CAutoTrackDoc serialization

void CAutoTrackDoc::Serialize(CArchive& ar)
{
     if (ar.IsStoring())
     {
           // TODO: add storing code here
     }
```

```
    else
    {
        // TODO: add loading code here
    }
}

/////////////////////////////////////////////////////////////////////////////
// CAutoTrackDoc diagnostics

#ifdef _DEBUG
void CAutoTrackDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CAutoTrackDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG


/////////////////////////////////////////////////////////////////////////////
// CAutoTrackDoc commands


/////////////////////////////////////////////////////////////////////////////
// Start capture video

void CAutoTrackDoc::OnCapture()
{
    // TODO: Add your command handler code here
    m_pCaptureth=(CaptureTh*)AfxBeginThread(RUNTIME_CLASS(CaptureTh),THREAD_
PRIORITY_NORMAL,0,CREATE_SUSPENDED);
   m_pCaptureth->SetOwner(this);
   m_pCaptureth->ResumeThread();
}

/////////////////////////////////////////////////////////////////////////////
// Stop capture video

void CAutoTrackDoc::OnStopcapture()
{
    // TODO: Add your command handler code here
    if(m_pCaptureth!=NULL)
   {
    m_pCaptureth->KillThread();
    m_pCaptureth=NULL;
   }
   else
    AfxMessageBox("Now, capture thread doesn't run");

}

/////////////////////////////////////////////////////////////////////////////
// Start tracking

void CAutoTrackDoc::OnTrack()
{
    // TODO: Add your command handler code here

m_pTrackingTh=(TrackingTh*)AfxBeginThread(RUNTIME_CLASS(TrackingTh),THREAD_PRI
ORITY_NORMAL,0,CREATE_SUSPENDED);
   m_pTrackingTh->SetOwner(this);
   m_pTrackingTh->ResumeThread();
```

```
}

////////////////////////////////////////////////////////////////////////////
// Stop tracking

void CAutoTrackDoc::OnStoptrack()
{
      // TODO: Add your command handler code here
   if(m_pTrackingTh!=NULL)
   {
      m_pTrackingTh->KillThread();
      m_pTrackingTh=NULL;
   }

}

////////////////////////////////////////////////////////////////////////////
// Check disk space

void CAutoTrackDoc::OnCheckdisk()
{
      // TODO: Add your command handler code here

   char driveroot[10]="c:\\";
   DWORD
sectorspercluster,bytespersector,numberoffreeclusters,totalnumberofclusters;
   BOOL canCheck;


canCheck=GetDiskFreeSpace(driveroot,&sectorspercluster,&bytespersector,&number
offreeclusters,&totalnumberofclusters);

   if (canCheck==TRUE)
   {
      char data[1000];
      int i=0;

      i+=sprintf(data+i,"SectorsPerCluster : %d \n",sectorspercluster);
      i+=sprintf(data+i,"BytesPerSector    : %d \n",bytespersector);
      i+=sprintf(data+i,"TotalFreeClusters : %d \n",numberoffreeclusters);
      i+=sprintf(data+i,"TotalFreeSpace    : %d
Kbytes\n",(long)(sectorspercluster*bytespersector*numberoffreeclusters)/1000);
      AfxMessageBox(data);
   }
   else
   {
      AfxMessageBox("Error invalid drive",MB_OK|MB_ICONEXCLAMATION);
   }

}

////////////////////////////////////////////////////////////////////////////
// Set video format

void CAutoTrackDoc::OnSetformat()
{
      // TODO: Add your command handler code here

   // Disable previous callback function for change format
   capSetCallbackOnFrame(hWndC,NULL);

   CAPDRIVERCAPS CapDrvCaps;

   capDriverGetCaps(hWndC,&CapDrvCaps,sizeof(CAPDRIVERCAPS));

   if (CapDrvCaps.fHasDlgVideoFormat)
```

```
    {
       capDlgVideoFormat(hWndC);
    }


    CView* pView;
    CWnd*  pFrame;
    POSITION pos;
    pos=GetFirstViewPosition();
    pView=(CView*)GetNextView(pos);
    pFrame=pView->GetParentFrame();

    CAPSTATUS CapStatus;

    // get status of capture window
    capGetStatus(hWndC,&CapStatus,sizeof(CAPSTATUS));

::MoveWindow(hWndC,0,GetSystemMetrics(SM_CYICON),(int)CapStatus.uiImageWidth,(
int)CapStatus.uiImageHeight,FALSE);
    CRect client,window;
    pFrame->GetClientRect(&client);
    pFrame->GetWindowRect(&window);
    client.right =client.left+window.Width() -client.Width()
+(int)CapStatus.uiImageWidth;
    client.bottom=client.top +window.Height()-
client.Height()+(int)CapStatus.uiImageHeight+GetSystemMetrics(SM_CYICON)+GetSy
stemMetrics(SM_CYSIZE);
    pFrame->MoveWindow(client.left,client.top,client.right,client.bottom,TRUE);
    pFrame->CenterWindow(NULL);

    // Disable previous allocate memmory block
    free(pbFrame1);
       free(pbFrame2);
       free(pbVideoFormat);
       free(pbResultFrame);

    // Set callback function again
    BOOL bSetCallbackStatus;

    bSetCallbackStatus=capSetCallbackOnFrame(hWndC,VideoFrameCallback);
       if (bSetCallbackStatus==FALSE)
       {
          AfxMessageBox("Error, can't set callback function");
    }

    // Access to videoformat header for image changed
       pbVideoFormat=(BYTE*)malloc(capGetVideoFormatSize(hWndC)*sizeof(BYTE));
       if (pbVideoFormat==NULL)
       {
             AfxMessageBox("Can't allocate memory for 'pbVideoFormat' ");
             exit(1);
       }

       lpbInfoHeader=(LPBITMAPINFOHEADER)pbVideoFormat;
       capGetVideoFormat(hWndC,lpbInfoHeader,capGetVideoFormatSize(hWndC));
       pRGB=(RGBQUAD*)(pbVideoFormat+lpbInfoHeader->biSize);

       // Allocate memory blocks and handle error for allocate memory
       pbBuffer=(BYTE*)malloc(lpbInfoHeader->biWidth*lpbInfoHeader->biHeight);
       if (pbBuffer==NULL)
       {
             AfxMessageBox("Can't allocate memory for 'pbBuffer' ");
             exit(1);
       }

       pbFrame1=(BYTE*)malloc(lpbInfoHeader->biWidth*lpbInfoHeader->biHeight);
```

```
	if (pbFrame1==NULL)
	{
		AfxMessageBox("Can't allocate memory for 'pbFrame1' ");
		exit(1);
	}

	pbFrame2=(BYTE*)malloc(lpbInfoHeader->biWidth*lpbInfoHeader->biHeight);
	if (pbFrame2==NULL)
	{
		AfxMessageBox("Can't allocate memory for 'pbFrame2' ");
		exit(1);
	}

	pbResultFrame=(BYTE*)malloc(lpbInfoHeader->biWidth*lpbInfoHeader-
	>biHeight);
	if (pbFrame2==NULL)
	{
		AfxMessageBox("Can't allocate memory for 'pbResultFrame' ");
		exit(1);

	}

	nHeight=lpbInfoHeader->biHeight;
	nWidth=lpbInfoHeader->biWidth;
	nRefXc=(int)(nWidth/2);

}

////////////////////////////////////////////////////////////////////////////
// Check time and date

void CAutoTrackDoc::OnChecktime()
{
	// TODO: Add your command handler code here

	CTime Time;

	Time=CTime::GetCurrentTime();
	CString strTime=Time.Format("The current date is: %d %B %Y\n The current
	time is: %H:%M");

	AfxMessageBox(strTime,MB_OK|MB_ICONEXCLAMATION);

}

////////////////////////////////////////////////////////////////////////////
// Test capture gray scale image

void CAutoTrackDoc::OnTestCaptureGrayImage()
{
	// TODO: Add your command handler code here

	ReadRefFrame();
	ToGrayScale(pbFrame1);
	RGB2GrayScale(pbFrame1);

}

////////////////////////////////////////////////////////////////////////////
// Read reference frame

void CAutoTrackDoc::ReadRefFrame()
{
	capGrabFrame(hWndC);
	memcpy(pbFrame1,pbBuffer,sizeof(BYTE)*nWidth*nHeight);
	capPreview(hWndC,TRUE);
```

```
}

////////////////////////////////////////////////////////////////////////////
// Callback function

LRESULT CALLBACK CAutoTrackDoc::VideoFrameCallback(HWND hWnd,LPVIDEOHDR
lpVHdr)
{
      DWORD i;

      for (i=0; i<(DWORD)(nWidth*nHeight); i++)
      {
            pbBuffer[i]=lpVHdr->lpData[i];
      }

      return 0;

}

////////////////////////////////////////////////////////////////////////////
// Check video format

void CAutoTrackDoc::OnInfo()
{
      // TODO: Add your command handler code here

      char data[500];
      int  index=0;

      index+=sprintf(data+index,"nBitPerPixel : %d\n",lpbInfoHeader-
>biBitCount);
      index+=sprintf(data+index,"nHeight : %d\n",nHeight);
      index+=sprintf(data+index,"nWidth : %d\n",nWidth);
      index+=sprintf(data+index,"nRefXc : %d\n",nRefXc);
      AfxMessageBox(data);

}

////////////////////////////////////////////////////////////////////////////
// Transform to gray scal

void CAutoTrackDoc::ToGrayScale(BYTE* pbGrayFrame)
{
      BYTE bBuffer;
      DWORD i;

      for (i=0; i<(DWORD)lpbInfoHeader->biWidth*lpbInfoHeader->biHeight; i++)
      {
            bBuffer=(BYTE)
            ((0.30f*(float)(pRGB[pbGrayFrame[i]].rgbRed  ))+
             (0.59f*(float)(pRGB[pbGrayFrame[i]].rgbGreen))+
             (0.11f*(float)(pRGB[pbGrayFrame[i]].rgbBlue )));
            pbGrayFrame[i]=bBuffer;
      }

}

////////////////////////////////////////////////////////////////////////////
// Save gray scale image file

void CAutoTrackDoc::RGB2GrayScale(BYTE* pbGrayFrame)
{

      BITMAPFILEHEADER bmfilehd;
      RGBQUAD rgbtable[256];
```

```
    for(int a=0;a<256;a++)
    {
       rgbtable[a].rgbBlue=a;
       rgbtable[a].rgbGreen=a;
       rgbtable[a].rgbRed=a;
       rgbtable[a].rgbReserved=0;
    }

    lpbInfoHeader->biClrUsed=256;
    lpbInfoHeader->biClrImportant=256;
    bmfilehd.bfType=0x4d42;
    bmfilehd.bfSize=0;
    bmfilehd.bfReserved1=0;
    bmfilehd.bfReserved2=0;
    bmfilehd.bfOffBits=sizeof(BITMAPFILEHEADER);

    CFile bmp;
    CString PathName;
    CFileDialog* FSaveDlg;

    FSaveDlg = new CFileDialog(FALSE,"BMP","",
                           OFN_OVERWRITEPROMPT|OFN_HIDEREADONLY,
                           "BMP (*.bmp)|*.bmp|All Files(*.*)|*.*||");

    if(FSaveDlg->DoModal()==IDOK)
    {
       PathName = FSaveDlg->GetPathName();

       bmp.Open(PathName,CFile::modeCreate|CFile::modeWrite,NULL);
       bmp.Write(&bmfilehd,sizeof(BITMAPFILEHEADER));
       bmp.Write(lpbInfoHeader,sizeof(BITMAPINFOHEADER));
       bmp.Write(&rgbtable,256*sizeof(RGBQUAD));
       bmp.Write(pbGrayFrame,lpbInfoHeader->biWidth*lpbInfoHeader->biHeight);

    }

    delete FSaveDlg;
}

////////////////////////////////////////////////////////////////////////////
// Test compare two image

void CAutoTrackDoc::OnCompare()
{
       // TODO: Add your command handler code here

    ReadRefFrame();
    ToGrayScale(pbFrame1);

    AfxMessageBox("To read second frame");

    ReadSecondFrame();
    ToGrayScale(pbFrame2);

    CompareImage();

    RGB2GrayScale(pbResultFrame);

    nXc=FindCentroidOfImage();


    char data[500];
    int  index=0;

    index+=sprintf(data+index,"nBitPerPixel : %d\n",lpbInfoHeader->biBitCount);
    index+=sprintf(data+index,"nHeight : %d\n",nHeight);
```

```
    index+=sprintf(data+index,"nWidth : %d\n",nWidth);
    index+=sprintf(data+index,"nXc : %d\n",nXc);
    AfxMessageBox(data);


}

////////////////////////////////////////////////////////////////////////////
// Compare two image

void CAutoTrackDoc::CompareImage()
{
    DWORD i;
    BYTE pBuf;

        for (i=0; i<(DWORD)(nWidth*nHeight); i++)
        {
              pBuf=255-abs(pbFrame1[i]-pbFrame2[i]);

        if ( pBuf> 200)
        {
           pbResultFrame[i]=255;
        }
        else
        {
           pbResultFrame[i]=0;
        }
    }
}

////////////////////////////////////////////////////////////////////////////
// Read second frame to compare

void CAutoTrackDoc::ReadSecondFrame()
{
        DWORD i;

        capGrabFrame(hWndC);

        for (i=0; i<(DWORD)(nWidth*nHeight); i++)
        {
              pbFrame2[i]=pbBuffer[i];
        }

        capPreview(hWndC,TRUE);


}

////////////////////////////////////////////////////////////////////////////
// Find centroid of result image

LONG CAutoTrackDoc::FindCentroidOfImage()
{
        LONG Xc,Yc;
        DWORD nArea;
        DWORD Xco,Yco;

        Xc=0;
        Yc=0;
        nArea=0;

        for (Yco=0; Yco<(DWORD)abs(nHeight); Yco++)
              for (Xco=0; Xco<(DWORD)nWidth; Xco++)
              {
                    if (pbResultFrame[Yco*nWidth+Xco]<100)
```

```
                {
                        Xc=Xc+Xco;
                        //Yc=Yc+Yco;
                        nArea++;
                }
        }

    if (Xc==0)
    {
        Xc=(LONG)(nWidth/2);
        //Yc=(LONG)nHeight/2;
        nArea=1;
    }

    Xc=(LONG)(Xc/nArea);

    return Xc;
}

////////////////////////////////////////////////////////////////////////////
// Threshold image

void CAutoTrackDoc::Threshold(BYTE* pbBuffFrame)
{
   DWORD Xco,Yco;


   for (Yco=0; Yco<(DWORD)abs(nHeight); Yco++)
        for (Xco=0; Xco<(DWORD)nWidth; Xco++)
        {
           if (pbBuffFrame[Yco*nWidth+Xco]>125)
           {
                pbBuffFrame[Yco*nWidth+Xco]=255;
           }
        else
        {
           pbBuffFrame[Yco*nWidth+Xco]=0;
        }
    }

}

////////////////////////////////////////////////////////////////////////////
// Find maximum pixel value

BYTE CAutoTrackDoc::FindMaxPixelValue(BYTE* pbBuffFrame)
{
   DWORD Xco,Yco;
   BYTE MAX;

   MAX=0;

   for (Yco=0; Yco<(DWORD)abs(nHeight); Yco++)
   {
        for (Xco=0; Xco<(DWORD)nWidth; Xco++)
        {
           if (pbBuffFrame[Yco*nWidth+Xco]>MAX)
           {
                MAX=pbBuffFrame[Yco*nWidth+Xco];
           }
        }
   }

   return MAX;

}
```

```
////////////////////////////////////////////////////////////////////////////
// Test threshold image and save to file

void CAutoTrackDoc::OnCapthres()
{
	// TODO: Add your command handler code here

	ReadRefFrame();
	ToGrayScale(pbFrame1);
	Threshold(pbFrame1);
	RGB2GrayScale(pbFrame1);

}

////////////////////////////////////////////////////////////////////////////
// Initial communication port2, must be initial before run tracking

BOOL CAutoTrackDoc::InitRS232()
{

	BOOL		fSuccess;
	COMMCONFIG CC;
	CView*		p_View;
	CWnd*		p_frame;
	POSITION	pos;
	DWORD		dwSize;

	hCOM2 = CreateFile("COM2",
					GENERIC_READ | GENERIC_WRITE,
					0,	/* comm devices must be opened
	w/exclusive-access */
					NULL, /* no security attrs */
					OPEN_EXISTING, /* comm devices must use
OPEN_EXISTING */
					FILE_FLAG_OVERLAPPED, /* overlapped I/O */
					NULL  /* hTemplate must be NULL for comm
devices */
				);

	if (hCOM2 == INVALID_HANDLE_VALUE)
	{
		AfxMessageBox("ERROR, can't create handle for COM2");
		return FALSE;
	}

	o.hEvent = CreateEvent( NULL,	// no security
				TRUE,	// explicit reset req
				FALSE,	// initial event reset
				NULL ) ; // no name

	if (o.hEvent==NULL)
	{
		AfxMessageBox("ERROR, can't create handle for EVENT");
		return FALSE;
	}

	pos=GetFirstViewPosition();
	p_View=(CView*)GetNextView(pos);
	p_frame=p_View->GetParentFrame();

	dwSize=sizeof(COMMCONFIG);
	GetCommConfig(hCOM2,&CC,&dwSize );
	CommConfigDialog("COM2",p_frame->GetSafeHwnd(),&CC);
	fSuccess=SetCommConfig(hCOM2,&CC,dwSize);
	if (!fSuccess)
```

```
      {
            AfxMessageBox("ERROR, can't open dlg");
            return FALSE;
      }


      return TRUE;
}

//---------------------------------------------------------------------------
//  BOOL CAutoTrackDoc::WriteCommBlock(LPSTR lpByte)
//
//  Description:
//     Writes a block of data to the COM port
//
//  Parameters:
//     BYTE *pByte
//           pointer to data to write to port
//
//  Win-32 Porting Issues:
//     - WriteComm() has been replaced by WriteFile() in Win-32.
//     - Overlapped I/O has not been implemented.
//
//---------------------------------------------------------------------------

BOOL CAutoTrackDoc::WriteCommBlock(LPSTR lpByte)
{

      BOOL        fWriteStat ;
      DWORD       dwBytesWritten ;

      fWriteStat = WriteFile(hCOM2,lpByte,1,&dwBytesWritten,NULL) ;
      if (!fWriteStat)
      {
            AfxMessageBox("Can't write COM2");
            return (FALSE);
      }
      else
      {
            //AfxMessageBox("Write COM2 success");
      }

      return (TRUE);

}

//---------------------------------------------------------------------------
//  int CAutoTrackDoc::ReadCommBlock(LPSTR lpszBlock, int nMaxLength )
//
//  Description:
//     Reads a block from the COM port and stuffs it into
//     the provided buffer.
//
//     int nMaxLength
//        max length of block to read
//
//  Win-32 Porting Issues:
//     - ReadComm() has been replaced by ReadFile() in Win-32.
//     - Overlapped I/O has been implemented.
//
//---------------------------------------------------------------------------

int CAutoTrackDoc::ReadCommBlock(LPSTR lpszBlock, int nMaxLength )
{
      BOOL       fReadStat;
```

```
        COMSTAT     ComStat ;
        DWORD       dwLength;
        DWORD       dwErrorFlags;

        // only try to read number of bytes in queue
        ClearCommError(hCOM2, &dwErrorFlags, &ComStat ) ;
        dwLength = min( (DWORD) nMaxLength, ComStat.cbInQue ) ;

        if (dwLength > 0)
        {
              fReadStat = ReadFile( hCOM2,lpszBlock,dwLength,&dwLength,NULL) ;
              if (!fReadStat)
              {
                    AfxMessageBox("Can't read from com2");
              }
        }
        else
        {
              AfxMessageBox("Not avaialbe data on com2");
        }

        return ( dwLength ) ;

}

////////////////////////////////////////////////////////////////////////////
// Close connection from rs232

BOOL CAutoTrackDoc::CloseConnection()
{
        // purge any outstanding reads/writes and close device handle

        if( ( PurgeComm(hCOM2, PURGE_TXABORT | PURGE_RXABORT |
                              PURGE_TXCLEAR | PURGE_RXCLEAR ) )&&
         ( CloseHandle(hCOM2) )
          )
                return TRUE;

        else
                return FALSE;

}


////////////////////////////////////////////////////////////////////////////
// Handle function of menu initial comm2

void CAutoTrackDoc::OnInitcom2()
{
        // TODO: Add your command handler code here
    BOOL fSuccess;

    fSuccess=InitRS232();
        if (!fSuccess)
    {
        AfxMessageBox("ERROR,Initialize commport2 failed");
    }
    else
    {
        AfxMessageBox("Initial success");
    }

}
```

```
void CAutoTrackDoc::OnTest232()
{
    // TODO: Add your command handler code here
    char m=20;

    DWORD dwLength;
    indata=0;

    PurgeComm(hCOM2,PURGE_RXCLEAR);

    WriteCommBlock(&m);

    do
    {
       ReadFile( hCOM2,&indata,1,&dwLength,&o);
    }
    while(indata==0);


    char data[1000];
    int i=0;

    i+=sprintf(data+i,"Data : %c \n",indata);
    AfxMessageBox(data);


}

void CAutoTrackDoc::OnCapcolor()
{
       // TODO: Add your command handler code here

}

void CAutoTrackDoc::OnUpdateStopcapture(CCmdUI* pCmdUI)
{
       // TODO: Add your command update UI handler code here
   pCmdUI->Enable(m_pCaptureth!=NULL);
}

void CAutoTrackDoc::OnUpdateStoptrack(CCmdUI* pCmdUI)
{
       // TODO: Add your command update UI handler code here
       pCmdUI->Enable(m_pTrackingTh!=NULL);
}
```

โปรแกรมภาษาแอสเซมบลี้ตระกูล MCS-51

```
;FILENAME       PROJ.ASM
;DESCRIPTION    CONTROL STEPPING MOTOR
;EXCITATION     TWO-STEP EXCITATION
;TX-RX MODE     MODE1(8 BIT UART) BAUDRATE = 9600 BIT/S
;HARDWARE       STEPPING MOTOR
;ASSEMBLER      SXA51
;SUBJECT        PROJECT I&II

;************************** PROGRAM ******************************

                        ORG 0000H

                    MOV   R2,#99H
                    MOV   P1,R2
                    MOV   PCON,#00H
                    MOV   SCON,#50H
                    MOV   TMOD,#20H
                    MOV   TH1,#0FDH
                    SETB  TR1

    NUMSTEP:        JNB   RI,NUMSTEP
                    CLR   RI
                    MOV   A,SBUF
                    MOV   R3,A
      TURN:         JNB   RI,TURN
                    CLR   RI
                    MOV   A,SBUF

          ; CHOOSE THE DIRECTION OF ROTATE

     CASE1:         CJNE  A,#04BH,CASE2
                    LCALL LEFT          ;ROTATE LEFT
                    LCALL TX
                    SJMP  NUMSTEP

     CASE2:         CJNE  A,#04DH,NUMSTEP
                    LCALL RIGHT         ;ROTATE RIGHT
                    LCALL TX
                    SJMP  NUMSTEP

          ; STEPPING MOTOR CONTROL PART

      LEFT:         MOV   A,R3
                    SUBB  A,#01H
                    JBC   CY,EXIT1
     LEFT1:         MOV   A,R2
                    RL    A
                    MOV   R2,A
                    ANL   A,#0FH      ;MAKE 4 BIT VALUE
                    LCALL SEND_PORT1
                    DJNZ  R3,LEFT1    ;UNTIL COUNTER=0
     EXIT1:         RET

     RIGHT:         MOV   A,R3
                    SUBB  A,#01H
                    JBC   CY,EXIT2
    RIGHT1:         MOV   A,R2
                    RR    A
                    MOV   R2,A
                    ANL   A,#0FH      ;MAKE 4 BIT VALUE
                    LCALL SEND_PORT1
                    DJNZ  R3,RIGHT1   ;UNTIL COUNTER=0
     EXIT2:         RET
```

```
          ; SEND DATA TO PORT1
          ; MAKE PULSE SIGNAL WHICH F=134 HZ
SEND_PORT1:       MOV   P1,A     ;START TO ROTATE
                  MOV   R1,#0AH ;LOAD APPROPRIATE VALUE TO ROTATE MOTOR
     DELAY:       MOV   R0,#83
       DEL:       NOP
                  NOP
                  DJNZ  R0,DEL
                  DJNZ  R1,DELAY
                  RET

          ; SEND INFORMATION BACK TO RS-232

        TX:       MOV   A,#05H  ; SEND ASCII"05H"TO PC
                  MOV   SBUF,A
      WAIT:       JNB   TI,WAIT
                  CLR   TI
                  RET

                      END
```

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลงได้ก็ด้วยความช่วยเหลือจากหลาย ๆ ฝ่าย ซึ่งทางคณะผู้จัดทำขอแสดงความขอบคุณเอาไว้ ณ. ที่นี้ ดังมีรายนามดังต่อไปนี้

1. ผศ. ดร. ไกรสิน ส่งวัฒนา ที่กรุณาเอื้อเฟื้ออุปกรณ์ในการทำปริญญานิพนธ์ อาทิเช่น กล้องวีดีโอ และอื่น ๆ รวมทั้งคำแนะนำต่าง ๆ
2. คุณเทอดศักดิ์ (พี่ต้น ป.โท) ที่กรุณาสละเวลาให้คำปรึกษาเกี่ยวกับแนวทางการเขียนโปรแกรม
3. อาจารย์ นภัทร สระเอี่ยม ที่กรุณาให้คำแนะนำในการเขียนโปรแกรมส่วนที่ทำหน้าที่ติดต่อกับพอร์ทอนุกรมบนระบบปฏิบัติการวินโดวส์
4. คุณณัฐพล จินดา ที่ให้ยืมอุปกรณ์ (มอเตอร์) ตัวอย่างที่ใช้ในการทดลอง
5. คุณเดโช อินทรรุจิกุล ที่ให้คำปรึษาเกี่ยวกับการเขียนโปรแกรมภาษาแอสเซมบลี้
6. อาจารย์ ทรงชัย วีระทวีมาศ ที่ให้คำปรึษาเกี่ยวกับการใช้งานสเต็ปปิ้งมอเตอร์

และสุดท้ายนี้ที่จะขาดเสียไม่ได้ก็คือ ขอขอบคุณบิดา- มารดา ผู้มีพระคุณ

หนังสืออ้างอิง

1. **สมยศ อุณณะปิยะ** , การใช้งานไมโครคอนโทรลเลอร์ตระกูล MCS-51 ; คณะวิศวกรรมศาสตร์ ส.จ.ล
2. **Randy Crane** , "A simplified approach to image processing classical and modern technique in C" ; Prentice-hall 1997
3. **Ioannis Pitas** , "Digital Image Processing Algorithm" ; Prentice-hall 1993
4. **Charlez Petzoid** , "Windows 95 Programming" ; Microsoft press 1996
5. **สุนทร วิทูสุรพจน์** , "การโปรแกรมภาษาแอสเซมบลี้ของไมโครคอนโทรลเลอร์ตระกูล MCS-51" ; ซีเอ็ดยูเคชั่น 1994
6. **ไกรวุฒิ โรจน์ประเสริฐสุด** ., "บอร์ดควบคุมสเต็ปปิ้งมอเตอร์ด้วยเครื่องพีซี" ; วารสารเซมิคอนดัคเตอร์ ฉบับที่ 146 (ก.พ. - มี.ค) 1995