หุ่นยนต์เตะฟุตบอล

SOCCER ROBOT

นายมณฑล ฉ่ำทรัพย์
นายกานต์ เงินเรืองนิรันดร์
นายภูรี เทียมใส

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมเครื่องกล
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2550

หุ่นยนต์เตะฟุตบอล

SOCCER ROBOT

โดย

นายมณฑล น่ำทรัพย์

นายกานต์ เงินเรืองนิรันดร์

นายภูรี เทียมใส

อาจารย์ที่ปรึกษา

ดร. ณัฐวุฒิ เดไปวา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมเครื่องกล

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2550

ปริญญานิพนธ์ปีการศึกษา 2550

ภาควิชา วิศวกรรมเครื่องกล

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง หุ่นยนต์เตะฟุตบอล

SOCCER ROBOT

ผู้จัดทำ

1. นายมณฑล ฉ่ำทรัพย์ รหัสประจำตัว 48015388
2. นายกานต์ เงินเรืองนิรันดร์ รหัสประจำตัว 48015412
3. นายภูรี เทียมใส รหัสประจำตัว 48015448

อาจารย์ที่ปรึกษา

(ดร. ณัฐวุฒิ เดไปวา)

หุ่นยนต์เตะฟุตบอล

นายมณฑล ฉ่ำทรัพย์ 48015388
นายกานต์ เงินเรืองนิรันดร์ 48015412
นายภูรี เทียมใส 48015448
ดร. ณัฐวุฒิ เดไปวา อาจารย์ที่ปรึกษา

บทคัดย่อ

โครงงานนี้เป็นการศึกษาถึงการออกแบบสร้างและควบคุมหุ่นยนต์เตะฟุตบอลให้ทำงานได้อย่างอัตโนมัติด้วยไมโครคอนโทรลเลอร์ ซึ่งในการศึกษาถึงการทำงานของหุ่นยนต์เตะฟุตบอลก็เป็นอีกทางหนึ่งที่จะทำให้เข้าใจถึงการควบคุมการทำงานของหุ่นยนต์ โดยในโครงงานนี้การออกแบบโครงสร้างของหุ่นยนต์ ตัวหุ่นยนต์ ล้อ และอุปกรณ์อื่นๆ ของหุ่นยนต์ โดยจุดประสงค์สำคัญของโครงงานนี้ คือ การมุ่งเน้นให้หุ่นยนต์นั้นสามารถเคลื่อนที่เข้าจุดที่ต้องการได้ โดยเป็นไปแบบอัตโนมัติ เริ่มจากการรับภาพจากกล้องไปประมวลผลที่คอมพิวเตอร์ และ ส่งสัญญาณไปยังหุ่นยนต์ ในโครงงานนี้จะเป็นการพัฒนาจากหุ่นยนต์ต้นแบบของหุ่นยนต์เตะฟุตบอลในปีการศึกษา 2549

SOCCER ROBOT

Mr.Monthol Chamsab	48015388
Mr.Kant Ngoenrueangniran	48015412
Mr.Puree Thiemsai	48015448
Dr.Nattawut Depaiwa	Advisor

Abstract

This project aim to comprehend robot mechanic through studying 'Robocup' by using mechanical discipline to design hardware. The project frame is to design the robot structure. Beside the movement system, the researchers also study about kicking system that use solenoid mechanism and for the dribbling mechanism we design the system to be able to calculate the best position for dribbling. For processing system, we use microcontroller system to process the signal from computer. This robot project is going to be prototype.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้จะไม่สามารถเสร็จลุล่วงไปได้ด้วยดี ถ้าหากปราศจากคำแนะนำ และความร่วมมือจากหลาย ๆ ฝ่ายด้วยกัน คือ ดร.ณัฐวุฒิ เดไปวา ผศ.ดร.อุนนัต พินโสภณ อาจารย์ที่ปรึกษาปริญญานิพนธ์ ที่ต้องกล่าวถึงเพราะเป็นส่วนสำคัญที่ทำให้ปริญญานิพนธ์นี้เสร็จลงได้ด้วยดี ที่ให้ความเอาใจใส่ และช่วยเหลือมาโดยตลอด ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก

ท้ายที่สุด คณะผู้จัดทำขอขอบพระคุณ คณาจารย์และบุคลากร ของภาคเครื่องกล ที่ให้การปรึกษาและความช่วยเหลือมาโดยตลอด ขอขอบพระคุณมา ณ ที่นี้

นายมณฑล ฉ่ำทรัพย์

นายกานต์ เงินเรืองนิรันดร์

นายภูรี เทียมใส

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญรูปภาพ

หน้าที่

สารบัญรูปภาพ (ต่อ)

บทที่ 1

บทนำ

1.1 ความสำคัญและความเป็นมา

ในปัจจุบันนี้หุ่นยนต์ที่สามารถปฏิบัติงานได้อย่างอัตโนมัติ ได้เริ่มเข้ามามีบทบาทอย่างมากในชีวิตประจำวันทั้งในด้านการอำนวยความสะดวกต่าง ๆ และมีประโยชน์อย่างมากในด้านอุตสาหกรรม โดยหุ่นยนต์ทั่วไปจะทำงานได้ก็ต้องมีผู้เขียนโปรแกรมการทำงานให้ เมื่อเราต้องการศึกษาและทดลองเกี่ยวกับหุ่นยนต์ เราก็จำเป็นต้องมีความรู้ในด้านการเขียนโปรแกรมโดยในโครงการนี้ได้ใช้โปรแกรม Visual Basic ในการเขียนโปรแกรม

การรับค่าจากผู้ใช้ คอมพิวเตอร์นั้นเราสามารถรับค่าได้หลายทาง เช่น เมาส์ แป้นพิมพ์ และอื่นๆ โดยนำค่าเหล่านั้นมาวิเคราะห์อีกทีหนึ่งถึงความต้องการของผู้ใช้จะทำให้การทำงานมีค่าความแม่นยำมากยิ่งขึ้นในการใช้คอมพิวเตอร์แทนคนนั้นในงานที่ซ้ำๆ หรือต้องการความแม่นยำโดยคนเราไม่สามารถทำงานได้แต่ว่าคอมพิวเตอร์ทำแทนได้ ทั้งนี้ก็จำเป็นจะต้องมีอีกฝ่ายหนึ่งที่ต้องผลิตโปรแกรมขึ้น ในการผลิตโปรแกรมนี้จะต้องให้ผู้ใช้สามารถใช้งานในลักษณะที่ง่ายแต่การใช้ และต้องมีความผิดพลาดน้อยที่สุดจึงจำเป็นจะต้องวิเคราะห์ให้ละเอียดด้วย

1.2 วัตถุประสงค์ของโครงงาน

เพื่อศึกษาการทำงานและนำไปประยุกต์ใช้ของหุ่นยนต์อัตโนมัติ โดยใช้ความรู้ทางด้าน การเขียนโปรแกรมประยุกต์ใช้งานด้วย โปรแกรมภาษาวิชวลเบสิกและการประยุกต์ใช้งานของไมโครคอนโทรลเลอร์ (Microcontroller) พร้อมด้วยการเขียนโปรแกรมติดต่อกับ พอร์ต USB และนำค่าจากการควบคุมมาประมวลผลและก็ส่งออกไปยังพอร์ตอนุกรมเพื่อสั่งให้ตัวหุ่นทำงานตามที่ต้องการได้

1.3 ขอบเขตของโครงงาน

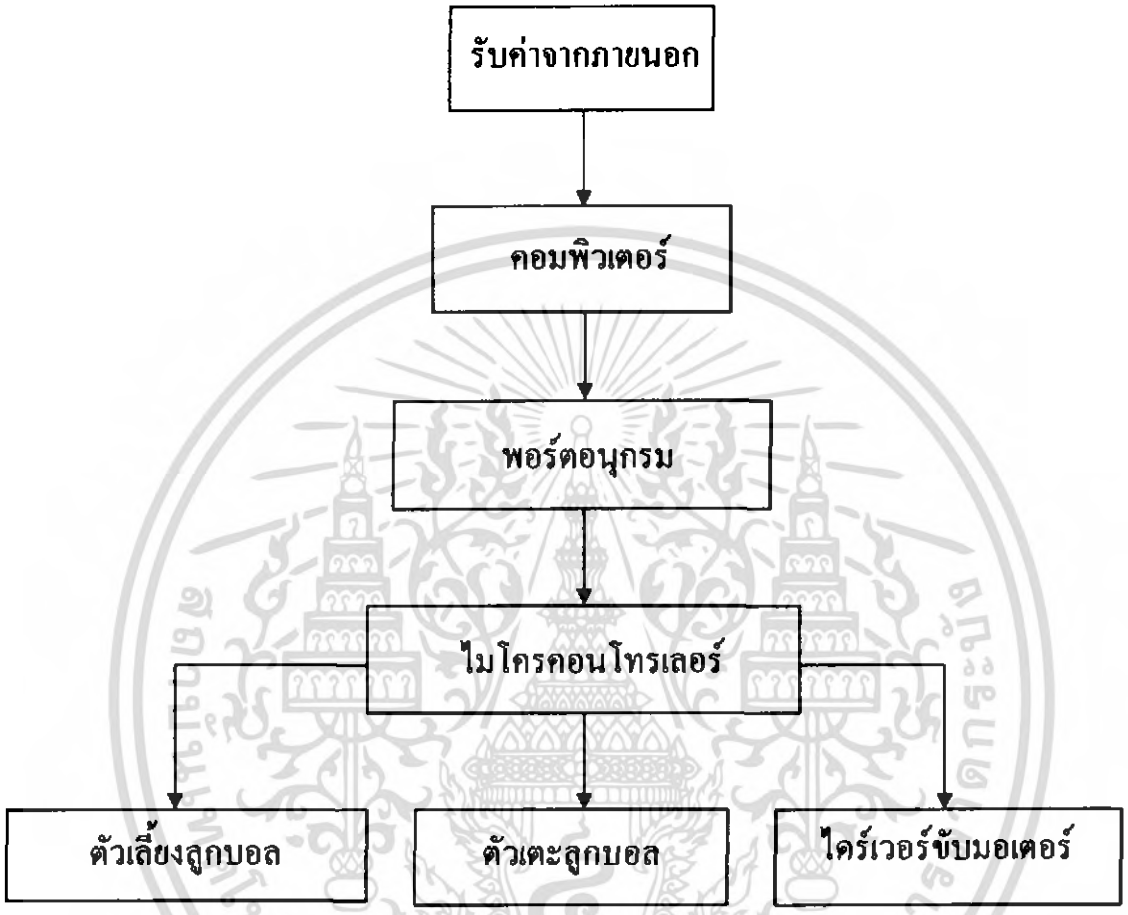
เนื้อหาของปริญญานิพนธ์นี้ จะเริ่มการศึกษาการทำงานของหุ่นยนต์แบบอัตโนมัติ โดยใช้สัญญาณภาพจากกล้องส่งมายังคอมพิวเตอร์ เพื่อคำนวณหาตำแหน่งพิกัดของตัวหุ่นจากภาพที่ได้รับจากกล้อง นำค่านั้นไปคำนวณการเคลื่อนที่ของตัวหุ่นแล้วส่งค่าออกไปที่พอร์ตอนุกรม (Serial port) และใช้ชุดรับส่งไร้สายในการส่งสัญญาณไปที่หุ่นยนต์จากนั้น Microcontroller จะรับค่าที่ส่งมาจากคอมพิวเตอร์ก็จะทำการแปลงความหมายของค่าที่ส่งมา เพื่อสั่งให้มอเตอร์เคลื่อนที่ไปยังตำแหน่งที่ต้องการ และได้ทำการพัฒนาจนสามารถประมวลผลหาตำแหน่งได้อย่างแม่นยำและหุ่นยนต์สามารถเคลื่อนที่ได้อย่างอัตโนมัติและเป็นทีม

1.4 ขั้นตอนการดำเนินโครงงาน

1. ศึกษาหลักการทำงานของกล้องและตัวรับส่งสัญญาณแบบไร้สายและไมโครคอนโทรลเลอร์
2. ออกแบบตัวหุ่นยนต์ให้มีความลงตัวกว่าเดิมและสร้างตัวหุ่นยนต์เตะฟุตบอลให้ครบทีม
3. ศึกษาโปรแกรมวิชวลเบสิก และสร้างโปรแกรมที่ใช้ในการคำนวณหาตำแหน่งพิกัดจากสัญญาณภาพที่ส่งจากกล้องมายังคอมพิวเตอร์
4. ศึกษาโปรแกรมภาษาซี และสร้างโปรแกรมที่ใช้กับไมโครคอนโทรลเลอร์
5. ศึกษาและสร้างไมโครคอนโทรลเลอร์ที่จะใช้ในการควบคุมโปรแกรมที่ใช้ในการกำหนดการเคลื่อนที่ของตัวหุ่นเมื่อได้รับค่าต่างๆ จากคอมพิวเตอร์
6. ติดตั้งและทำการทดลอง
7. สรุปผลการทดลอง
8. แนวทางแก้ไขและพัฒนา
9. นำเสนอผลงาน

บทที่ 2

อุปกรณ์และทฤษฎีที่ใช้ในหุ่นยนต์เตะฟุตบอล



รูปที่ 2-1 Block diagram แสดงหลักการทำงานของหุ่นยนต์เตะฟุตบอล

2.1 ไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์ คือ ไอซีหน่วยประมวลผลกลาง (Central Processing Unit) ประเภทหนึ่งที่สามารถรับสัญญาณข้อมูลจากอุปกรณ์ภายนอก เพื่อนำมาคำนวณ ตัดสินใจและส่งสัญญาณออกไปยังอุปกรณ์ภายนอกเพื่อสั่งให้ทำงานตามโปรแกรม และข้อมูลที่รับเข้ามา มีคุณสมบัติความเป็น Single Chip คือ สามารถทำงานได้โดยตัวไอซีเอง ไม่ต้องต่อวงจรเพิ่มหรือต่อเพิ่มน้อยที่สุด ประโยชน์ที่ได้รับก็คือ การออกแบบวงจร

ทำได้ง่ายขึ้น ใช้อุปกรณ์ประกอบวงจรน้อยกว่าเดิม พื้นที่วงจรรวมมีขนาดเล็กลงเป็นอย่างมาก กินไฟเลี้ยงวงจรน้อยลง ทำให้ต้นทุนการผลิตและต้นทุนการทำงานลดต่ำลง นอกจากนี้ การบันทึกโปรแกรมที่เขียนลงในตัวไมโครคอนโทรลเลอร์ก็สามารถทำได้ง่ายและรวดเร็ว โดยเลือกรุ่นที่มีแฟลชเมโมรีในตัวที่สามารถเขียนโปรแกรมลงไปได้ ทำให้ไม่ต้องต่ออีพรอมเพื่อเก็บโปรแกรม

ในโครงงานนี้ใช้ไมโครคอนโทรลเลอร์ตระกูล AVR ของบริษัท ATMEL เบอร์ ATMEGA128 แบบ TQFP 64 PIN โดยสามารถต่อสัญญาณนาฬิกาได้ 16 MHz มีหน่วยความจำแบบ FLASH 128 kbyte หน่วยความจำโปรแกรม 4 kbyte และ RAM 4 kbyte

2.1.1 ความหมายของไมโครคอนโทรลเลอร์

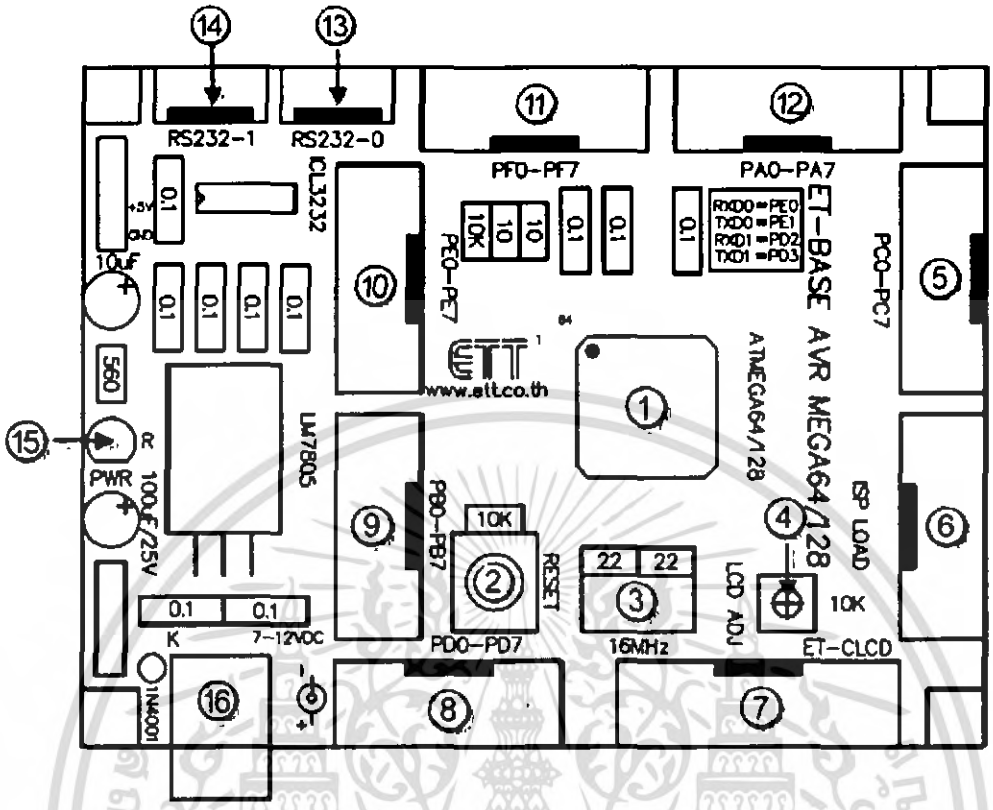
ไมโครคอนโทรลเลอร์ (Microcontroller) เป็นชื่อของอุปกรณ์อิเล็กทรอนิกส์แบบหนึ่งที่รวมเอาหน่วยประมวลผล หน่วยคำนวณทางคณิตศาสตร์และลอจิก วงจรรับสัญญาณอินพุต วงจรขับสัญญาณเอาท์พุต หน่วยความจำ วงจรกำเนิดสัญญาณนาฬิกาไว้ด้วยกัน ทำให้สามารถนำไปใช้งานแทนวงจรอิเล็กทรอนิกส์ที่ซับซ้อนได้เป็นอย่างดี ช่วยลดจำนวนอุปกรณ์และขนาดของระบบ ในขณะที่มีขีดความสามารถสูงขึ้น ภายใต้งบประมาณที่เหมาะสม ดังนั้น ไมโครคอนโทรลเลอร์จึงเป็นอุปกรณ์ที่ใช้ในการควบคุม โดยที่สามารถเขียนโปรแกรมเพื่อกำหนดรูปแบบการควบคุมได้อย่างอิสระ

2.1.2 คุณสมบัติของไมโครคอนโทรลเลอร์ตระกูล AVR อนุกรม ATMEGAxx

ไมโครคอนโทรลเลอร์ตระกูล AVR อนุกรม ATMEGAxx มีคุณสมบัติดังนี้คือ

- เป็นไมโครคอนโทรลเลอร์ที่ใช้เอ็มซียูขนาด 8 บิต
- ภายในมีหน่วยความจำโปรแกรมเป็นแบบแฟลชสามารถลบและเขียนใหม่ได้เป็นแสนครั้ง
- หน่วยความจำข้อมูลพื้นฐานเป็นหน่วยความจำแบบแรม และมีหน่วยความจำแบบอีพรอม
- ขาพอร์ตเป็นแบบสองทิศทาง สามารถใช้งานเป็นได้ทั้งอินพุต-เอาท์พุต จำนวน 6 พอร์ต
- มีวงจรสื่อสารอนุกรม
- มีไทเมอร์/เคาน์เตอร์ขนาด 16 บิตจำนวน 2 ช่อง และ 8 บิต จำนวน 2 ช่อง
- สามารถรองรับแหล่งกำเนิดอินเทอร์รัป
- สามารถขยายหน่วยความจำภายนอกเพิ่มเติมได้
- มีวงจรกำเนิดสัญญาณนาฬิกาอยู่ภายในชิป
- มีวงจรสื่อสารอนุกรมแบบ SPI สำหรับในอนุกรม

2.1.3 โครงสร้างบอร์ดไมโครคอนโทรลเลอร์



รูปที่ 2-2 แสดงโครงสร้างบอร์ดไมโครคอนโทรลเลอร์ AVR ATMEGAxx

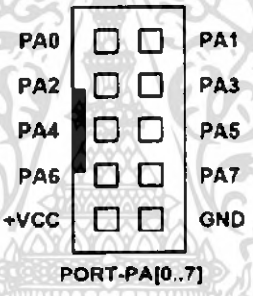
- **หมายเลข 1** คือ เอ็มพียู เบอร์ ATMEGA128 ซึ่งเป็นเอ็มพียูตระกูล AVR จาก ATMEL
- **หมายเลข 2** คือ สวิทช์รีเซทใช้สำหรับรีเซทการทำงานของเอ็มพียู
- **หมายเลข 3** คือ คริสตอลสัญญาณนาฬิกา 16 MHz
- **หมายเลข 4** คือ ตัวต้านทานสำหรับปรับค่าความสว่างให้ LCD
- **หมายเลข 5** คือ พอร์ต C มีขนาด 8 บิต คือ PC0-PC7
- **หมายเลข 6** คือ พอร์ต ISP LOAD ใช้สำหรับการดาวน์โหลด Hex File ให้กับเอ็มซียู
- **หมายเลข 7** คือ พอร์ต ET-CLCD สำหรับเชื่อมต่อกับ LCD ชนิด Character Type ซึ่งใช้การเชื่อมต่อแบบ 4 บิต
- **หมายเลข 8** คือ พอร์ต D มีขนาด 8 บิต คือ PD0-PD7
- **หมายเลข 9** คือ พอร์ต B มีขนาด 8 บิต คือ PB0-PB7
- **หมายเลข 10** คือ พอร์ต E มีขนาด 8 บิต คือ PE0-PE7
- **หมายเลข 11** คือ พอร์ต F มีขนาด 8 บิต คือ PF0-PF7

- **หมายเลข 12** คือ พอร์ต A มีขนาด 8 บิต คือ PA0-PA7
- **หมายเลข 13** คือ ข้อต่อ RS232 สำหรับใช้งานทั่วไป
- **หมายเลข 14** คือ ข้อต่อ RS232 สำหรับใช้งานทั่วไป
- **หมายเลข 15** คือ LED Power ใช้สำหรับแสดงสถานะของแหล่งจ่ายไฟ +5VDC
- **หมายเลข 16** คือ ขั้วต่อแหล่งจ่ายไฟสำหรับเลี้ยงวงจรของบอร์ด

2.1.4 ขั้วต่อสัญญาณของพอร์ตในไมโครคอนโทรเลอร์

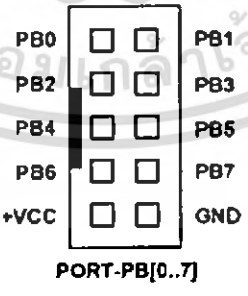
สำหรับขั้วต่อสัญญาณของพอร์ต I/O จาก MCU นั้นจะถูกออกแบบและจัดเตรียมไว้ผ่านทางขั้วต่อแบบ IDC-Header ขนาด 10 Pin (2X5) จำนวน 6 ชุด คือ PA, PB, PC, PD, PE, PF ตามลำดับ โดยที่ขั้วต่อสัญญาณแต่ละชุด จะประกอบไปด้วยสัญญาณของ I/O ที่เชื่อมต่อมาจากขาสัญญาณของ MCU โดยตรงทั้งหมด โดยจุดเชื่อมต่อกับสัญญาณภายนอกบอร์ดมีดังนี้

- ขั้วต่อแหล่งจ่ายไฟสำหรับเลี้ยงวงจรของบอร์ด
- ขั้วต่อ PORTA มีขนาด 8 Bit คือ PA0-PA7



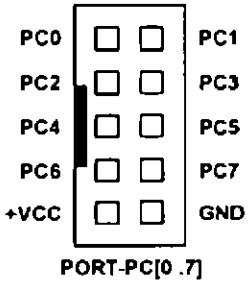
รูปที่ 2-3 แสดง PORTA มีขนาด 8 Bit คือ PA0-PA7

- ขั้วต่อ PORTB มีขนาด 8 Bit คือ PB0-PB7



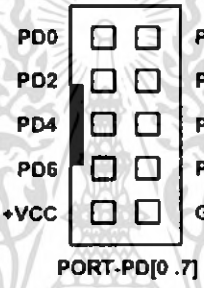
รูปที่ 2-4 แสดง PORTB มีขนาด 8 Bit คือ PB0-PB7

- ขั้วต่อ PORTC มีขนาด 8 Bit คือ PC0-PC7



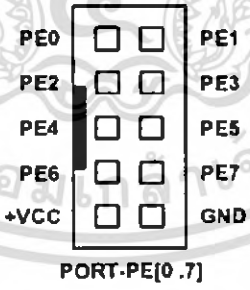
รูปที่ 2-5 แสดง PORTC มีขนาด 8 Bit คือ PC0-PC7

- ขั้วต่อ PORTD มีขนาด 8 Bit คือ PD0-PD7



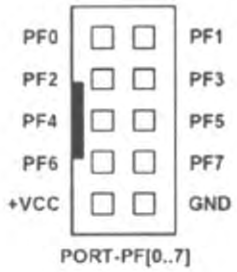
รูปที่ 2-6 แสดง PORTD มีขนาด 8 Bit คือ PD0-PD7

- ขั้วต่อ PORTE มีขนาด 8 Bit คือ PE0-PE7



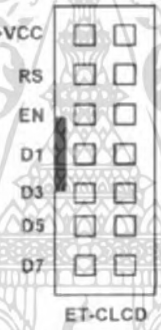
รูปที่ 2-7 แสดง PORTE มีขนาด 8 Bit คือ PE0-PE7

- ขั้วต่อ PORTF มีขนาด 8 Bit คือ PF0-PF7

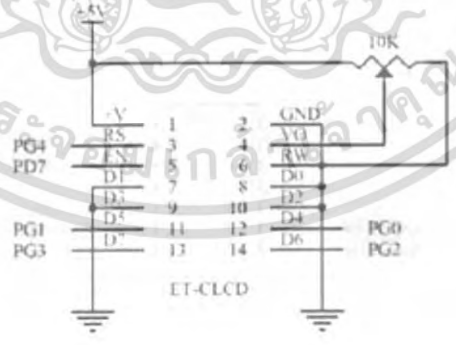


รูปที่ 2-8 แสดง PORTF มีขนาด 8 Bit คือ PF0-PF7

- ขั้วต่อ ET-CLCD สำหรับเชื่อมต่อกับ LCD ชนิด Character Type โดยใช้การเชื่อมต่อแบบ 4 บิต โดยสัญญาณที่ใช้เชื่อมต่อกับ LCD จะเป็นสัญญาณจากพอร์ต PG และ PD (PD7) โดยในการเชื่อมต่อสายสัญญาณจากขั้วต่อของพอร์ต LCD ไปยังจอแสดงผล LCD นั้นให้ยึดชื่อขาสัญญาณเป็นจุดอ้างอิง โดยให้ต่อสัญญาณที่มีชื่อตรงกันเข้าด้วยกันให้ครบทั้ง 14 เส้น

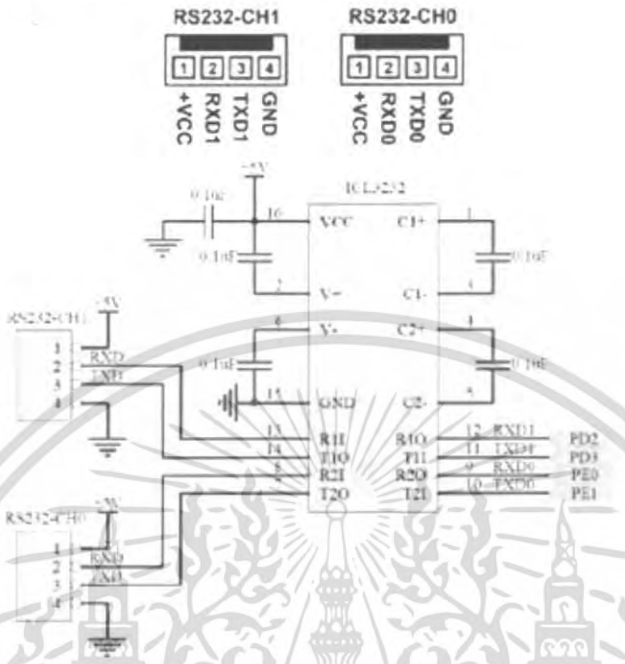


รูปที่ 2-9 แสดงขั้วต่อ ET-CLCD



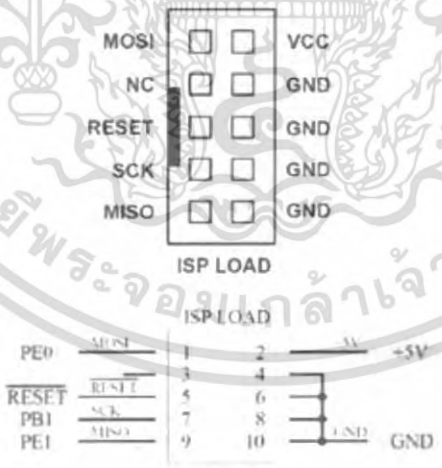
รูปที่ 2-10 แสดงวงจรขั้วต่อ ET-CLCD

- ขั้วต่อ RS232 จำนวน 2 ช่อง โดยเชื่อมต่อกับสัญญาณ PE0(RXD0) และ PE1(TXD0) จำนวน 1 ช่อง ส่วนที่เหลืออีก 1 ช่อง จะต่อกับสัญญาณ PD2(RXD1) และ PD3(TXD1) เพื่อให้ผู้ใช้สามารถต่อทดลองการติดต่อสื่อสาร RS232



รูปที่ 2-11 แสดงวงจรขั้วต่อ RS232

- ขั้วต่อ ISP LOAD ใช้สำหรับดาวน์โหลด Hex File ให้กับ MCU



รูปที่ 2-12 แสดงวงจรและขั้วต่อ ISP LOAD

2.1.5 การทำงานของพอร์ตในไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์ AVR มีพอร์ตให้ใช้งานทั้งสิ้น 6 พอร์ต คือ พอร์ต A จนถึง พอร์ต F แต่ละพอร์ตมีขนาด 8 บิต เป็นพอร์ตแบบ 2 ทิศทาง กล่าวคือ สามารถเป็นได้ทั้งอินพุตสำหรับรับสัญญาณข้อมูลเข้า และเอาต์พุตสำหรับส่งสัญญาณข้อมูลออก ทุกพอร์ตของไมโครคอนโทรลเลอร์ AVR

2.1.6 การใช้งานเป็นพอร์ตอินพุต

เนื่องจากพอร์ตทั้งหมดของไมโครคอนโทรลเลอร์ AVR สามารถเป็นได้ทั้งอินพุตและเอาต์พุต ดังนั้นจึงมีความจำเป็นอย่างยิ่งต้องทำความเข้าใจถึงการกำหนดลักษณะการทำงานให้แก่พอร์ตของไมโครคอนโทรลเลอร์ AVR

ในการกำหนดให้เป็นพอร์ตอินพุตต้องเริ่มต้นด้วยการเขียนข้อมูล "1" มาที่แต่ละบิตของพอร์ตที่ต้องการใช้งานเป็นอินพุต เพื่อหยุดการทำงานของเฟตที่ใช้ในการขับสัญญาณเอาต์พุตของบิตนั้นๆ ทำให้ขาสัญญาณของพอร์ตเชื่อมต่อเข้ากับวงจรพูลอัปภายในโดยตรง ส่งผลให้ขาพอร์ตนั้นมีลอจิกเป็น "1" สามารถรับสัญญาณลอจิก "0" จากอุปกรณ์ภายนอกได้ง่าย สัญญาณข้อมูลจากอุปกรณ์ภายนอกจะถูกส่งเข้ามาแล้วเก็บไว้ในวงจรบัฟเฟอร์ภายในพอร์ตแล้วรอให้ซีพียูมาอ่านค่าเข้าไป เมื่อเป็นเช่นนี้ อุปกรณ์ภายนอกที่เชื่อมต่อกับพอร์ตอินพุตของไมโครคอนโทรลเลอร์ AVR ควรกำหนดให้ทำงานในสภาวะลอจิก "0" จะดีและสะดวกที่สุด ซึ่งในปัจจุบันอุปกรณ์อินพุตที่เชื่อมต่อไมโครคอนโทรลเลอร์แทบทั้งหมดทำงานที่ลอจิก "0" แล้ว

2.1.7 การใช้งานเป็นพอร์ตเอาต์พุต

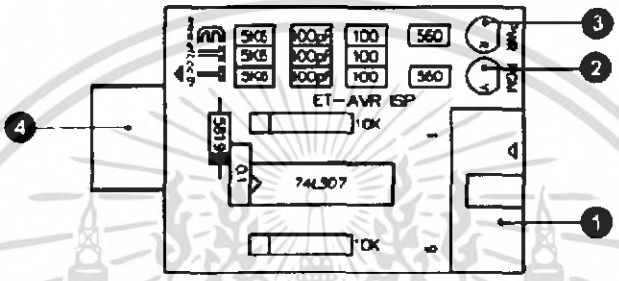
โดยปกติแล้ ขาพอร์ตจะกำหนดให้มีลักษณะเป็นเอาต์พุตอยู่แล้ว ดังนั้นจึงสามารถส่งข้อมูลออกไปได้อย่างง่ายดายและตรงไปตรงมา กล่าวคือ เมื่อต้องการส่งข้อมูล "0" ออกไปทางเอาต์พุตก็ให้เขียนข้อมูล "0" ไปยังวงจรแลตช์ ซึ่งก็จะส่งต่อไปยังเฟต ทำให้เฟตทำงาน ที่ขาพอร์ตที่กำหนดให้ทำงานก็จะเกิดลอจิก "0" ขึ้น ในทางตรงข้ามหากต้องการส่งข้อมูล "1" ออกไป ก็ให้เขียนข้อมูล "1" ไปยังวงจรแลตช์ วงจรขับก็จะหยุดทำงาน ทำให้ที่ขาพอร์ตเชื่อมต่อกับวงจรพูลอัปภายในเกิดเป็นลอจิก "1" ที่ขาพอร์ตนั้น ซึ่งจะคล้ายกับการกำหนดให้เป็นขาอินพุตมาก เพียงแต่แตกต่างกันที่กระบวนการในการเคลื่อนย้ายข้อมูล โดยถ้าเป็นอินพุตจะมีสัญญาณมาอ่านข้อมูลที่บัฟเฟอร์ แต่ถ้าเป็นเอาต์พุตจะไม่มีการอ่านข้อมูลที่บัฟเฟอร์แต่อย่างใดเว้นแต่ในกรณีที่ต้องการตรวจสอบข้อมูลที่ส่งออกมาทางเอาต์พุต

2.1.8 การดาวน์โหลด Hex File ให้กับ MCU

การดาวน์โหลด Hex File ให้กับ MCU นั้นจำเป็นจะต้องใช้ ET-AVR ISP หรือเครื่องโปรแกรมแบบ ISP อื่นๆ เช่น AVRISP ของ ATMEL เพื่อใช้ในการดาวน์โหลด Hex File ให้กับ MCU ตระกูล AVR ของ Atmel โดยใช้วิธีการแบบ Serial Programming ซึ่งการดาวน์โหลด Hex File ในกรณีที่ใช้ ET-AVR ISP จะ

กระทำผ่านทางพอร์ตขนานของคอมพิวเตอร์ โดยที่จะต้องใช้งานร่วมกับ ET-CAP10P ของอีทีที และ Software ที่ใช้ร่วมกับ ET-AVR ISP ก็คือ PonyProg2000 ซึ่ง PonyProg2000 เป็นโปรแกรม Download ข้อมูลแบบ HEX File ให้กับ CPU ตระกูล AVR โดยใช้วิธีการแบบ Serial Programming ซึ่งสามารถใช้งานกับบอร์ดตระกูล AVR ของ อีทีที ได้เป็นอย่างดี ซึ่งวิธีการใช้งานโปรแกรมโดยทั่วไปนั้น สามารถศึกษาได้จาก Help ของโปรแกรมได้เอง โดยในที่นี้จะขอแนะนำให้ทราบถึงวิธีการ Setup โปรแกรม PonyProg2000 เพื่อใช้งานกับบอร์ดตระกูล AVR ของ อีทีที ซึ่งสามารถใช้งานได้กับบอร์ดตระกูล AVR ทุกรุ่นของ อีทีที

2.1.9 โครงสร้างของบอร์ด ET-AVR ISP

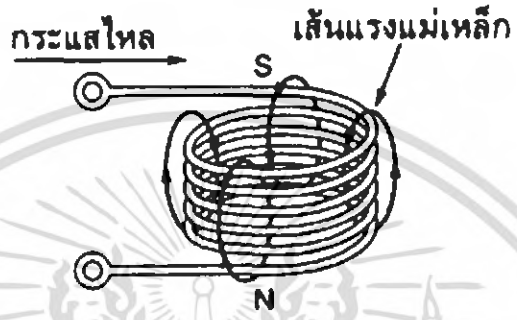


รูปที่ 2-13 แสดงโครงสร้างของบอร์ด ET-AVR ISP

- หมายเลข 1 คือ พอร์ตสำหรับเชื่อมต่อกับ ET-CAP10P ของอีทีที เพื่อโปรแกรม Hex File ให้กับ MCU
- หมายเลข 2 คือ LED PGM (สีเขียว) แสดงสถานะของการโปรแกรมหรือดาวน์โหลด Hex File ลง MCU
- หมายเลข 3 คือ LED PWR (สีแดง) แสดงสถานะของไฟเลี้ยงบอร์ด
- หมายเลข 4 คือ พอร์ตสำหรับเชื่อมต่อกับบอร์ด Target ซึ่งสามารถใช้โปรแกรม Hex File ให้กับบอร์ด ET-AVR STAMP ATmega128 โดยเสียบบอร์ด ET-AVR ISP เข้าที่ พอร์ต ET-PSPI ซึ่งมีการจัดเรียงขาสัญญาณดังรูป

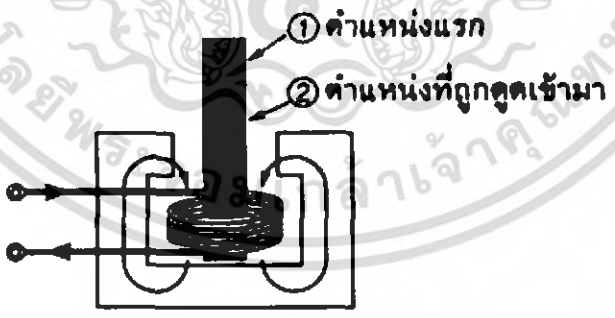
2.2 โซลินอยด์ไฟฟ้า

โซลินอยด์ไฟฟ้ามีหลักการทำงานดังนี้ คือ เมื่อมีกระแสไฟฟ้าไหลในขดลวดตัวนำใดๆก็ตาม จะก่อให้เกิดสนามแม่เหล็กขึ้นรอบๆตัวนำนั้น และหากนำเส้นลวดมาขดเป็นวงๆ หลายๆวง ก็จะเกิดลักษณะของขดลวดขึ้น โดยสนามแม่เหล็กที่เกิดจากขดลวดแต่ละขดจะอยู่ในทิศทางเสริมกัน และก่อกำเนิดเป็นเส้นแรงของสนามแม่เหล็กถาวรแท่งหนึ่ง ซึ่งพร้อมที่จะดูดสารแม่เหล็กทันที แต่เนื่องจากสภาพรอบๆ ขดลวดอาจเป็นอากาศ เส้นแรงแม่เหล็กจึงไม่เข้มข้นมากนัก



รูปที่ 2-14 แสดงถึงทิศทางของสนามแม่เหล็กที่เกิดขึ้นในขดลวดที่มีกระแสไหล

เพื่อไม่ให้สนามแม่เหล็กที่เกิดขึ้นกระจัดกระจาย จึงใส่แกนเหล็กอ่อนรูปตัวซีล้อมรอบๆขดลวดไว้ เพื่อให้สนามแม่เหล็กที่เกิดขึ้นกระจัดกระจายออกไป และเพื่อให้สนามแม่เหล็กมีความเข้มมากขึ้น จากจุดนี้หากนำเอาแกนกระทุ้ง (Plunger) มาใส่เข้าไปตรงกลางวงของขดลวดในตำแหน่งที่ 1 แกนกระทุ้งจะถูกดูด ให้ลึกเข้ามาจนสนิทในตำแหน่งที่ 2 ความสัมพันธ์ของแรงกับระยะช่วงชักของโซลินอยด์ ในช่วงชักไกลๆจะมีแรงน้อยมาก แต่ในทางตรงกันข้ามช่วงชักที่มีระยะใกล้ๆก็จะมีแรงมากขึ้นเป็นทวีคูณ



รูปที่ 2-15 แสดงถึงการเพิ่มเหล็กอ่อนเข้ามาเพื่อเพิ่มความเข้มของสนามแม่เหล็ก

2.2.1 ประเภทของโซลินอยด์ไฟฟ้า

โซลินอยด์ไฟฟ้าสามารถแบ่งได้เป็น 2 ประเภท คือ โซลินอยด์ที่ใช้ไฟฟ้ากระแสตรง และโซลินอยด์ที่ใช้ไฟฟ้ากระแสสลับ ข้อแตกต่างระหว่างโซลินอยด์ที่ใช้ไฟฟ้ากระแสตรง และโซลินอยด์ที่ใช้ไฟฟ้ากระแสสลับ คือ โซลินอยด์ที่ใช้ไฟกระแสตรงจะก่อให้เกิดกระแสที่ไหลในขดลวดค่อนข้างคงที่ไม่เปลี่ยนแปลง ไม่ว่าแกนกระทุ้งจะอยู่ในตำแหน่งใดก็ตาม แต่โซลินอยด์ที่ใช้ไฟกระแสสลับจะมีกระแสในขณะที่แกนกระทุ้งอยู่นอกขดลวดค่อนข้างสูง และเมื่อแกนกระทุ้งถูกดูดเข้ามาจนสุดขดลวด กระแสจะลดต่ำลง ด้วยเหตุนี้เองที่ทำให้ต้องระวังอย่าให้เกิดการกระทุ้งในโซลินอยด์ไฟสลับ เพราะจะทำให้เกิดกระแสมากๆไหลค้างอยู่ ทำให้ขดลวดร้อนขึ้น และ อาจจะไหม้เสียหายได้

2.2.2 การเลือกใช้โซลินอยด์ไฟฟ้า

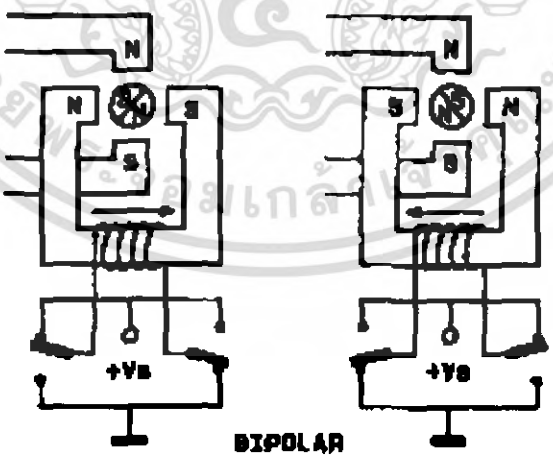
การเลือกใช้โซลินอยด์ไฟฟ้าควรพิจารณาถึงองค์ประกอบต่างๆ ดังนี้ คือ

1. แรงดันใช้งานซึ่งไม่ว่าจะเป็นไฟฟ้ากระแสตรง หรือไฟฟ้ากระแสสลับก็ตาม จำเป็นจะต้องดูความถี่ใช้งานให้ตรงตามความต้องการด้วย ซึ่งในการออกแบบและสร้างเครื่องเจาะพลาสติกเพื่อใช้เป็นแบบสำหรับทอพรมนี้ใช้โซลินอยด์ที่มีความถี่ 55 เฮิร์ซ

2. ช่วงชักในการใช้งาน (Operating Stroke) ของโซลินอยด์จะต้องเคลื่อนที่เป็นระยะทางเท่าใด มักจะกำหนดเป็นมิลลิเมตร และการสร้างตัวหุ่นนี้จะใช้โซลินอยด์ขนาดเล็กและมีแรงขับที่พอเหมาะด้วย

3 .เป็นการใช้งานอย่างต่อเนื่องหรือไม่ ซึ่งการใช้งานอย่างต่อเนื่องในที่นี้ หมายถึง การใส่แรงดันไฟเข้าขดลวดค้างไว้โดยขดลวดไม่ไหม้ หรือเป็นการใส่แรงดันแบบเป็นจังหวะๆ เพื่อใช้ในการเตะลูกฟุตบอลจะต้องเลือกคือต้องเล็กและมีแรงในการเตะที่แรงด้วย

2.3 มอเตอร์



รูปที่ 2-16 *แสดงวงจรของมอเตอร์เมื่อทิศทางการหมุนที่ต่างกัน*

2.3.1 ประเภทของมอเตอร์

มอเตอร์ที่นิยมนำมาใช้ในงานคอนโทรลปัจจุบันจะแบ่งออกได้เป็น 3 ประเภท คือ หนึ่ง สเต็ปเปอร์มอเตอร์ (Stepper Motor) สอง เซอร์โวมอเตอร์ (Servo Motor) และสาม มอเตอร์ไฟฟ้ากระแสตรง (DC Motor) ซึ่งจะกล่าวถึงดังต่อไปนี้

2.3.1.1 สเต็ปเปอร์มอเตอร์

สเต็ปเปอร์มอเตอร์ เป็นมอเตอร์ที่หมุนตามจำนวนองศาหรือตามจำนวนรอบที่ต้องการ ทั้งนี้ความละเอียดของการหมุนจะขึ้นอยู่กับจำนวนองศาต่อหนึ่งจังหวะการหมุน เนื่องจากการหมุนของมอเตอร์ชนิดนี้มีการหมุนเป็นจังหวะ จึงเรียกว่า สเต็ปเปอร์มอเตอร์ และที่นิยมใช้จะเป็นชนิดที่เรียกว่าสเต็ปเปอร์มอเตอร์แบบยูนิโพล่าร์ (Uni-polar Stepper Motor)

2.3.1.2 เซอร์โวมอเตอร์

เซอร์โวมอเตอร์เป็นมอเตอร์ที่ประกอบไปด้วยชุดเกียร์ (Gear) มอเตอร์กระแสตรง (DC Motor) และส่วนควบคุมอิเล็กทรอนิกส์ ที่รวมอยู่ภายในตัวมอเตอร์ เซอร์โวมอเตอร์จะทำงานได้ด้วยสัญญาณพัลส์ที่มีคาบความกว้างอยู่ระหว่าง 1 มิลลิวินาที ถึง 2 มิลลิวินาที (พัลส์บวกหรือลอจิก 1) โดยส่งพัลส์ดังกล่าวห่างกันเป็นระยะเวลา 20 มิลลิวินาที (พัลส์ลบหรือ ลอจิก 0) การส่งสัญญาณพัลส์ดังกล่าวมีผลให้มอเตอร์หมุน โดยทิศทางการหมุนนั้นจะขึ้นอยู่กับความกว้างของพัลส์บวก

2.3.1.3 มอเตอร์ไฟฟ้ากระแสตรง

มอเตอร์ไฟฟ้ากระแสตรงเป็นมอเตอร์ที่ขับเคลื่อนด้วยไฟฟ้ากระแสตรง โดยควบคุมความเร็วของมอเตอร์ไฟฟ้ากระแสตรงได้ด้วยการจ่ายกระแสไฟฟ้าเป็นช่วงเวลา แต่ไม่สามารถที่จะควบคุมทิศทางการหมุนของมอเตอร์ได้ การที่จะควบคุมทิศทางการหมุนของมอเตอร์จะต้องใช้ไอซีมาช่วยในการขับเคลื่อน โดยใช้หลักการขับเคลื่อนแบบเอชบริดร์ (H-Bridge)

2.3.2 การเลือกใช้มอเตอร์

จะเห็นได้ว่ามอเตอร์ทั้ง 3 ประเภทนี้ ต่างก็มีข้อดีข้อเสียต่างกันออกไป การที่เราจะตัดสินใจเลือกใช้มอเตอร์แบบใดนั้นคงต้องพิจารณาจากจุดประสงค์หลักในการใช้งาน สำหรับในหุ่นยนต์แตะฟุตบอลนี้จะเลือกใช้มอเตอร์ชนิดมอเตอร์ไฟฟ้ากระแสตรงเพราะต้องการความเร็วและการหมุนที่ราบเรียบ

2.4 อุปกรณ์รับและส่งข้อมูล

ในการติดต่อสื่อสารและรับส่งข้อมูลกันระหว่างหุ่นยนต์กับคอมพิวเตอร์จะต้องมีสื่อกลางในการรับและส่งข้อมูลซึ่งกันและกัน ดังนั้นจึงต้องมีอุปกรณ์ที่ช่วยในการรับและส่งข้อมูล อุปกรณ์รับและส่งข้อมูลก็จะแบ่งออกเป็น 2 ประเภทใหญ่ๆได้ดังนี้ คือ แบบใช้สายสัญญาณในการรับส่งข้อมูล และแบบไม่ใช้สายสัญญาณในการรับส่งข้อมูลหรือก็คือแบบไร้สายนั่นเองในหุ่นยนต์จะใช้การรับส่งข้อมูลแบบไร้สาย ดังนั้นจะพูดถึงเฉพาะแบบไร้สายเท่านั้น

2.4.1 ชุดรับส่งข้อมูล ET-RF24G V1.0

ลักษณะโดยทั่วไปของ ET-RF24G V1.0 เป็นชุด Signal Converter สำหรับใช้แปลงสัญญาณระหว่าง RS232 และ RF-Wireless โดยในโหมดการทำงานของการส่งข้อมูล (Transmitter) จะทำหน้าที่รอรับข้อมูลจากพอร์ตสื่อสารอนุกรม RS232 จากขา RX แล้วแปลงเป็นสัญญาณความถี่ (GFSK) ส่งออกไปในอากาศ และในทางกลับกันในโหมดการทำงานแบบรับ (Receiver) ชุด ET-RF24G V1.0 ก็จะทำหน้าที่คอยตรวจจับข้อมูลที่อยู่ในรูปของสัญญาณความถี่ (GFSK) จากด้าน RF เพื่อแปลงกลับเป็นข้อมูลแบบ RS232 ส่งออกไปทางขา TX ได้ด้วย

ซึ่งจะเห็นได้ว่าชุดแปลงสัญญาณ ET-RF24G V1.0 นั้น สามารถนำไปต่อใช้งานร่วมกับพอร์ตสื่อสารอนุกรม แบบ RS232 เพื่อใช้งานในลักษณะของการสื่อสารอนุกรมแบบไร้สาย (Wireless Transceiver) ได้โดยตรง โดยจะมีข้อดีกว่า คือ สามารถรับส่งข้อมูลกันได้ในระยะทางที่ไกลกว่า RS232 หลายเท่าตัว และประการสำคัญ คือ ไม่จำเป็นต้องใช้สายสัญญาณที่เป็นตัวนำสัญญาณทางไฟฟ้าในการสื่อสารข้อมูลกันทำให้สามารถเปลี่ยนแปลงหรือเคลื่อนย้ายชุดรับส่งข้อมูลได้ตลอดเวลา

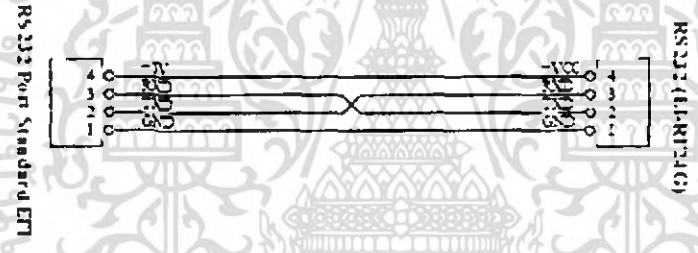
ซึ่งถ้าเป็นการรับส่งข้อมูลด้วยระบบ RS232 แบบที่ใช้สายสัญญาณนั้นจะเกิดความยุ่งยากในการติดตั้งสายสัญญาณเป็นอย่างมากแต่อย่างไรก็ตามการรับส่งข้อมูลโดยใช้อากาศเป็นตัวกลางในการสื่อสารนั้น ก็มีข้อจำกัดบางประการเหมือนกัน โดยเฉพาะอย่างยิ่งเรื่องความน่าเชื่อถือของข้อมูลที่รับส่งกัน ซึ่งมีโอกาสผิดพลาดหรือสูญหายได้เหมือนกัน เนื่องจากในการลำเลียงข้อมูลนั้นไม่ได้ใช้สายสัญญาณเป็นตัวกลางในการรับส่งข้อมูล แต่ใช้อากาศเป็นตัวกลางในการรับส่งข้อมูลแทน ซึ่งมีโอกาสที่ข้อมูลจะเกิดการรบกวนจากสัญญาณอื่นๆที่มีย่านความถี่ใกล้เคียงกันแล้วทำให้ข้อมูลผิดเพี้ยนไปได้บ้างเหมือนกัน ซึ่งระบบการจัดการข้อมูลของเครื่อง ET-RF24G V1.0 นั้น มีระบบการเข้ารหัสและถอดรหัสข้อมูลที่มีความน่าเชื่อถืออยู่ในเกณฑ์ที่จัดว่าดี โดยข้อมูลแต่ละ Byte ที่มีการรับส่งกันนั้นจะมีการตรวจสอบความถูกต้องของข้อมูลให้ด้วยแล้ว โดยข้อมูลที่รับได้จากด้าน RF นั้นรับประกันได้ว่าเป็นข้อมูลที่มีความถูกต้องแน่นอน แต่อย่างไรก็ตามการรับส่งข้อมูลนั้นมีโอกาสผิดพลาดในเรื่องของการสูญหายของข้อมูลบ้างเหมือนกัน เนื่องจากกลไกในการรับส่งข้อมูลของเครื่อง ET-RF24G V1.0 นั้น จะมีการตรวจสอบข้อมูลทุก Byte ที่รับได้จาก RF เสมอ ซึ่งถ้าพบว่ามีความผิดพลาดเกิดขึ้นจะทิ้งข้อมูล Byte นั้นไป ซึ่งผู้ใช้ควรมีกลไกในการตรวจสอบข้อมูลที่รับส่งกันว่า

ครบถ้วนหรือไม่ด้วย ซึ่งหากพบว่ามีการสูญหายของข้อมูลเกิดขึ้นก็ให้ร้องขอให้มีการส่งข้อมูลนั้นซ้ำ ใหม่อีกครั้งหนึ่ง ก็จะสามารถแก้ไขปัญหาดังกล่าวได้

2.4.2 Power Supply

สำหรับการต่อแหล่งจ่ายไฟให้กับเครื่อง ET-RF24G V1.0 นั้น จะสามารถเลือกต่อแหล่งจ่ายไฟให้กับตัวเครื่องได้ 2 ทางด้วยกัน โดยเครื่อง ET-RF24G V1.0 นั้นต้องการไฟเลี้ยงวงจร ซึ่งเป็นแหล่งจ่ายกระแสตรง ขนาดประมาณ +5VDC ถึง +9VDC โดยจุดเชื่อมต่อแหล่งจ่ายไฟของเครื่อง ET-RF24G V1.0 นี้ สามารถเชื่อมต่อได้ 2 จุดด้วยกัน โดยผู้ใช้สามารถเลือกต่อแหล่งจ่ายไฟให้กับเครื่อง ET-RF24G V1.0 จุดใดจุดหนึ่งก็ได้

ในกรณีที่นำเครื่อง ET-RF24G V1.0 ไปเชื่อมต่อกับบอร์ดไมโครคอนโทรลเลอร์รุ่นต่างๆของ อีทีที นั้นสามารถใช้แหล่งจ่ายไฟจากบอร์ดไมโครคอนโทรลเลอร์ เพื่อจ่ายให้กับตัวเครื่อง ET-RF24G V1.0 ได้ทันที โดยไม่ต้องใช้จ่ายไฟจากภายนอก เนื่องจากขั้วต่อสัญญาณ RS232 ของบอร์ดไมโครคอนโทรลเลอร์รุ่นต่างๆ ของ บริษัท อีทีที นั้นๆ ได้จัดเตรียมแหล่งจ่ายไฟขนาด +5V เตรียมไว้ให้ด้วยแล้ว โดยผู้ใช้เพียงแต่นำสายสัญญาณ RS232 ซึ่งทำการต่อสายสัญญาณครบทั้ง 4 เส้น ดังรูปมาเชื่อมต่อก็สามารถใช้งานได้แล้ว



รูปที่ 2.17 แสดงการต่อสัญญาณ RS232 เพื่อใช้แหล่งจ่ายไฟจากบอร์ดไมโครคอนโทรลเลอร์ของอีทีที

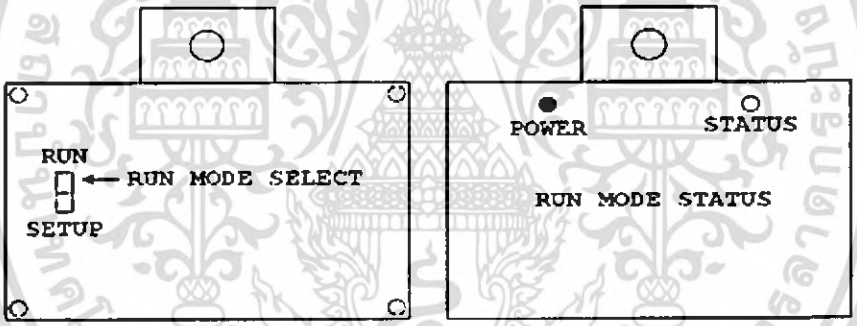
แต่สำหรับกรณีที่นำเครื่อง ET-RF24G V1.0 ไปต่อใช้งานกับอุปกรณ์อื่นๆที่ไม่ได้มีการจัดเตรียมจุดต่อไฟเลี้ยงไว้ให้ด้วย ผู้ใช้จำเป็นต้องจัดหา Adapter จ่ายไฟจากภายนอกมาต่อให้กับเครื่อง ET-RF24G V1.0 ต่างหากด้วย โดยให้เลือกแหล่งจ่ายไฟที่มีขนาดแรงดันไฟตรงประมาณ +5VDC และสามารถจ่ายกระแสได้ประมาณ 300 mA เป็นอย่างน้อย

สำหรับโหมดการทำงาน ET-RF24G V1.0 นั้นจะแบ่งออกเป็น 2 โหมด ด้วยกัน โดยการกำหนดโหมดการทำงานของ ET-RF24G V1.0 นั้นจะกระทำผ่าน Switch เลือกโหมด ซึ่งอยู่ด้านใต้กล่อง โดยการเลือกโหมดการทำงานนั้นจะต้องกระทำให้เสร็จเรียบร้อยก่อนการจ่ายไฟให้กับ ET-RF24G V1.0 ด้วยเสมอ เนื่องจากการทำงานของเครื่อง ET-RF24G V1.0 นั้นจะทำการตรวจสอบโหมดการทำงานของเครื่องจาก Switch เลือกโหมด เฉพาะในช่วงของการจ่ายไฟเลี้ยงให้เครื่องเริ่มต้นทำงานครั้งแรก (Power-ON) เท่านั้น ซึ่ง

การเปลี่ยนแปลงตำแหน่งการทำงานของ Switch เลือกโหมด หลังการทำงานจ่ายไฟให้กับ ET-RF24G V1.0 ไปแล้ว จะไม่มีผลต่อการทำงานของเครื่องแต่อย่างใด โดยการทำงานของเครื่อง ET-RF24G V1.0 นั้นจะมี LED แสดงสถานการณ์ทำงานของเครื่องจำนวน 2 หลอด คือ LED POWER ซึ่งเป็น LED สีแดง โดยที่ LED POWER นี้จะติดสว่างให้เห็นตลอดเวลาที่มีการจ่ายไฟเลี้ยงให้เครื่องทำงานอยู่ ส่วน LED อีกดวงหนึ่งนั้นจะเป็น LED สีเขียว ใช้แสดงสถานการณ์ทำงานของเครื่องซึ่งเรียกว่า LED STATUS นี้จะเกิดการกระพริบตามจังหวะของการรับส่งข้อมูลกันในแต่ละครั้งโดนสภาวะปรกตินั้น ถ้าเครื่องทำงานอยู่ใน RUN MODE หลอด LED STATUS จะดับอยู่ตลอดเวลาถ้าไม่มีการรับส่งข้อมูล แต่ถ้าเครื่องทำงานอยู่ใน SETUP MODE หลอด LED STATUS จะติดอยู่ตลอดเวลาถ้าไม่มีการรับส่งข้อมูลโดยโหมดการทำงาน ET-RF24G V1.0 จะมีอยู่ด้วยกัน 2 โหมด คือ

2.4.2.1 การใช้งานเครื่อง ET-RF24G V1.0 ใน RUN Mode

การใช้งานใน RUN Mode ซึ่งเป็นโหมดของการใช้งานตามปรกติของเครื่อง โดยเมื่อเครื่อง ET-RF24G V1.0 เข้าทำงานในโหมดนี้แล้ว จะสังเกตเห็นหลอดไฟแสดงสถานการณ์ทำงานของ LED STATUS จึงจะกระพริบตามจังหวะของการรับส่งข้อมูลนั้นๆแต่ถ้ายังไม่มีการรับส่งข้อมูลกัน LED STATUS จะดับอยู่ตลอดเวลา



รูปที่ 2.18 แสดง การเลือกโหมดการทำงานสำหรับใช้งานปรกติ (Run Mode)

สำหรับการทำงานใน Run Mode นั้นจะแบ่งลักษณะการทำงานออกเป็น 3 แบบด้วยกัน โดยลักษณะการทำงานนี้จะถูกกำหนดไว้แล้วใน Configuration ของเครื่องใน Step Mode ดังนั้นก่อนการใช้งานเครื่อง ในครั้งแรกจะต้องทำการกำหนดค่า Configuration ต่างๆให้เรียบร้อยเสียก่อน โดยเมื่อเครื่อง ET-RF24G V1.0 เริ่มต้นเข้าทำงานใน Run Mode แล้วจะทำการอ่านค่า Configuration ที่เก็บไว้ออกมา เพื่อใช้เป็นเงื่อนไขในการทำงานตามค่าที่ได้กำหนด โดยลักษณะการทำงานใน Run Mode แบ่งออกเป็นดังนี้

2.4.2.1.1 การทำงานแบบ RF Receive Only

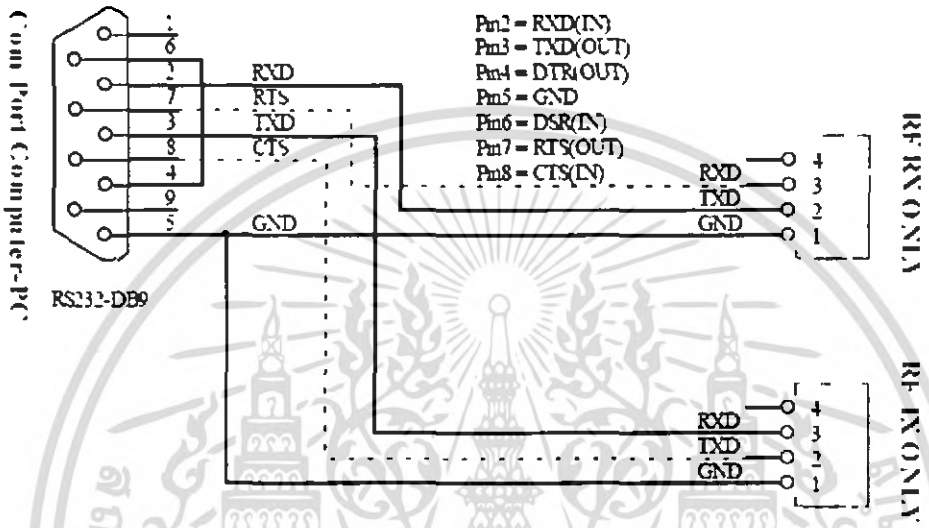
เป็นการทำงานแบบทิศทางเดียวโดยการทำงานในโหมดนี้ จะเป็นการรอรับข้อมูลความถี่แบบ GFSK จากด้าน RF แล้วเปลี่ยนเป็นข้อมูลอนุกรมส่งออกไปทางขา TX (Transmit) ของ RS232 โดยการทำงานจะวนรอบอยู่เช่นนี้ไปตลอด ซึ่งในการใช้เครื่อง ET-RF24G V1.0 ในโหมดนี้จะต้องนำสัญญาณ TX (Transmit) ไปต่อกับขาสัญญาณ RX (Receive) ของอุปกรณ์ด้านตรงข้าม (RS232 ของคอมพิวเตอร์ PC) โดยในโหมดนี้การทำงานของขาสัญญาณ RX ด้าน RS232 ของเครื่อง ET-RF24G V1.0 จะถูกเปลี่ยนหน้าที่เป็นสัญญาณ CTS (Clear To Send) สำหรับใช้ตรวจสอบความพร้อมในการส่งข้อมูลไปให้อุปกรณ์ด้านตรงกันข้าม ซึ่งในการใช้งานจะต้องนำสัญญาณนี้ไปต่อเข้ากับสัญญาณ RX ซึ่งในโหมดนี้เปรียบเสมือน CTS ว่ามีค่าเป็น "0" หรือไม่ โดยถ้าพบว่าเป็น "0" จึงจะส่งข้อมูลออกไปให้ทางขา TX แต่ถ้าพบว่าสถานะของสัญญาณดังกล่าวมีค่าเป็น "1" แสดงว่าอุปกรณ์ด้านตรงข้ามยังไม่พร้อมรับข้อมูลก็จะรอจนกว่าจะพบว่าสถานะของสัญญาณดังกล่าวมีค่าเป็น "0" จึงจะส่งข้อมูลออกไปให้ โดยเครื่อง ET-RF24G V1.0 จะสามารถจัดเก็บข้อมูลไว้ใน Buffer เพื่อรอการส่งได้สูงสุด 64 Byte เท่านั้น ซึ่งถ้าในระหว่างที่รอความพร้อมอยู่นั้นมีข้อมูลด้าน RF ส่งเข้ามาเกินกว่า 64 Byte จะทำให้ข้อมูลที่เกินมานั้นสูญหายไป

2.4.2.1.2 การทำงานแบบ RF Transmit Only

เป็นการทำงานแบบทิศทางเดียว โดยการทำงานในโหมดนี้จะมีลักษณะตรงกันข้าม RF Receive Only กล่าวคือ เครื่อง ET-RF24G V1.0 จะทำหน้าที่รอรับข้อมูลจากขา RX (Receive) ด้าน RS232 แล้วเปลี่ยนเป็นข้อมูลแบบ GFSK ส่งออกไปทางด้าน RF โดยใช้งานเครื่องในโหมดนี้จะต้องนำสัญญาณ TX (Transmit) ซึ่งเป็นขาส่งข้อมูลจาก RS232 ของอุปกรณ์ด้านตรงข้ามมาต่อเข้ากับขา RX (Receive) ของเครื่อง ET-RF24G V1.0 ส่วนขาสัญญาณ TX จะถูกเปลี่ยนหน้าที่เป็น RST (Ready To Send) เพื่อใช้แสดงสถานะความพร้อมในการรับข้อมูลจากด้าน RS232 ซึ่งในการใช้งานจะต้องนำสัญญาณ TX ซึ่งในขณะนี้เปรียบเสมือนกับ RST นำไปต่อเข้ากับสัญญาณ CTS (Clear To Send) ของอุปกรณ์ด้านตรงข้าม เพื่อใช้ในการตรวจสอบความพร้อมในการรับข้อมูล โดยอุปกรณ์ด้านตรงข้ามจะต้องทำการตรวจสอบสถานะของสัญญาณ RST นี้ เพื่อตรวจสอบความพร้อมในการรับข้อมูล โดยอุปกรณ์ด้านตรงข้ามจะต้องทำการตรวจสอบสถานะของสัญญาณ RST เพื่อตรวจสอบความพร้อมในการรับข้อมูลของเครื่อง ET-RF24G V1.0 ด้วยโดยถ้าเครื่อง ET-RF24G V1.0 พร้อมรับข้อมูลจาก RS232 มันจะส่งสัญญาณ RST ให้มีค่าเป็น "0" รอไว้และเมื่อใดก็ตามที่การรับข้อมูลทางด้านของ RS232 มีจำนวนข้อมูลที่ยังไม่สามารถเปลี่ยนเป็น GFSK เพื่อส่งออกไปทางด้าน RF ได้ทันจนเกือบจะเต็ม Buffer แล้วเครื่อง ET-RF24G V1.0 จะทำการส่งสัญญาณ RST ให้มีค่าเป็น "1" ออกไปบอกให้อุปกรณ์ด้านตรงข้ามทราบเพื่อจะได้หยุดการส่งข้อมูลออกมา โดยอุปกรณ์ด้านตรงข้ามจะต้องหยุดการส่งข้อมูล และจนกว่าสถานะของสัญญาณ RST จะกลับเป็น "0" จึงจะเริ่มต้นส่งขอมูลมาใหม่ซึ่งหลังจากที่เครื่อง ET-RF24G V1.0 ส่งสัญญาณ RST ด้วย "1" ออกไปแล้ว จะยังคงส่งข้อมูลได้เพิ่มเติมอีกไม่เกิน 16 Byte เท่านั้น ซึ่งถ้า

อุปกรณ์ด้านตรงข้ามยังส่งข้อมูลต่อเนื่องมาอีกจนเกินขนาดของ Buffer ที่เครื่อง ET-RF24G V1.0 จะรับไว้ได้ จะทำข้อมูลที่เกินมานั้นเกิดการเสียหายได้

โดยเราสามารถนำเครื่อง ET-RF24G V1.0 จำนวน 4 ชุด มาต่อใช้งานร่วมกัน เพื่อใช้งานในการส่งข้อมูลกันแบบ Full Duplex โดยแบ่งการใช้งานออกเป็น 2 ด้าน ต้นทาง และ ปลายทาง ด้านละ 2 ชุด โดยแต่ละด้านให้กำหนดหน้าที่การทำงานเป็น RF Receive Only 1 ชุด และ RF Transmit Only อีก 1 ชุด



รูปที่ 2.19 แสดงสายสัญญาณ RS232 เพื่อใช้กับ ET-RF24G ในโหมดรับและส่ง

2.4.2.1.3 การทำงานแบบ RF Auto Direction

เป็นการทำงานชนิด 2 ทิศทาง แบบ Half Duplex หรือผลัดกันรับผลัดกันส่ง ซึ่งสามารถรับส่งข้อระหว่างต้นทางและปลายทางได้ โดยใช้เครื่อง ET-RF24G V1.0 ด้านละ 1 ชุดเท่านั้น เพียงแต่การรับส่งข้อมูลแบบนี้จะไม่สามารถส่งข้อมูลสวนทางกันได้เหมือนกันแบบ Full Duplex แต่จะต้องใช้วิธีการผลัดกันรับข้อมูลและส่งข้อมูลแทน โดยเมื่อฝ่ายรับทำการรับข้อมูลจนครบแล้วจึงจะสลับหน้าที่เป็นฝ่ายส่งเพื่อส่งข้อมูลย้อนกลับไป

โดยในโหมดนี้เครื่อง ET-RF24G V1.0 จะทำหน้าที่เป็นฝ่ายรับและส่งข้อมูลแบบอัตโนมัติ โดยในสภาวะปรกติจะอยู่ในสภาวะของการรอรับข้อมูล ทั้งด้าน RF และ RS232 ทันที และในทำนองเดียวกันถ้าพบมีข้อมูลส่งเข้ามาทางด้าน RX ของ RS232 มันก็จะทำการรับข้อมูลนั้นจาก RS232 พร้อมกับเปลี่ยนทิศทางของอุปกรณ์ RF จากการรอรับข้อมูลให้ทำหน้าที่เป็นตัวส่งข้อมูลแทน เพื่อทำการส่งข้อมูลที่รับได้จาก RS232 ออกไปทาง RF ในทันที ซึ่งหลังจากที่เครื่อง ET-RF24G V1.0 ทำการสลับโหมดการทำงานของอุปกรณ์ด้าน RF จากการรอรับเป็นการส่งและทำการเริ่มต้นส่งข้อมูลออกไปทางด้าน RF เรียบร้อย มันจะวนกลับไปตรวจสอบการรับข้อมูลจากด้าน RS232 อีกว่ายังมีข้อมูลส่งเข้ามาอีกหรือไม่ ถ้าพบว่ายังมีข้อมูลส่งเข้ามีอีกก็จะทำการแปลงข้อมูลนั้นเพื่อส่งออกไปยังด้าน RF ต่อไปอีกจนกว่าการส่งข้อมูลด้าน RS232 จะสิ้นสุดลง ซึ่ง

ข้อมูลทางด้าน RS232 ที่ส่งเข้ามานั้นควรส่งอย่างต่อเนื่องโดยเมื่อเครื่อง ET-RF24G V1.0 ทำการส่งข้อมูลแต่ละ Byte ออกไปทางด้าน RF เรียบร้อยแล้วมันจะวนรอบรอรับข้อมูล Byte ถัดไปจาก RS232 ภายใน 2.5 mS ถ้าไม่พบข้อมูลส่งเข้ามาอีกภายในระยะเวลาดังกล่าวมันจึงจะทำการเปลี่ยนหน้าที่ของอุปกรณ์ด้าน RF ให้กลับมาทำหน้าที่เป็นการรับข้อมูลตามเดิม โดยในขณะที่อุปกรณ์ด้าน RF ถูกกำหนดให้เป็นฝ่ายส่งข้อมูลอยู่นั้นจะไม่สามารถทำการรับข้อมูลจาก RF ได้ ซึ่งถ้ามีการส่งข้อมูลเข้ามาในขณะนั้นก็จะไม่สามารถรับได้ โดยค่าเวลาที่ใช้สลับโหมดการทำงานจาก RF จากฝ่ายส่งข้อมูลให้เป็นฝ่ายรับข้อมูลนั้น จะมีค่าเป็น 2.5 mS ดังนั้นเมื่อฝ่ายรับสามารถข้อมูลได้ครบหมดแล้วก่อนที่จะทำการส่งข้อมูลเพื่อตอบกลับไปในฝ่ายตรงข้ามนั้น ควรทำการหน่วงเวลาไว้ไม่น้อยกว่า 3 mS นับจากรับข้อมูล Byte สุดท้ายได้เรียบร้อยแล้วจึงเริ่มต้นส่งข้อมูล Byte แรกย้อนกลับไปซึ่งถ้าฝ่ายรับทำการส่งข้อมูลตอบกลับไปยังฝ่ายตรงข้ามเร็วกว่านี้อาจทำให้ฝ่ายตรงข้ามไม่สามารถรับข้อมูล Byte แรกได้ทัน

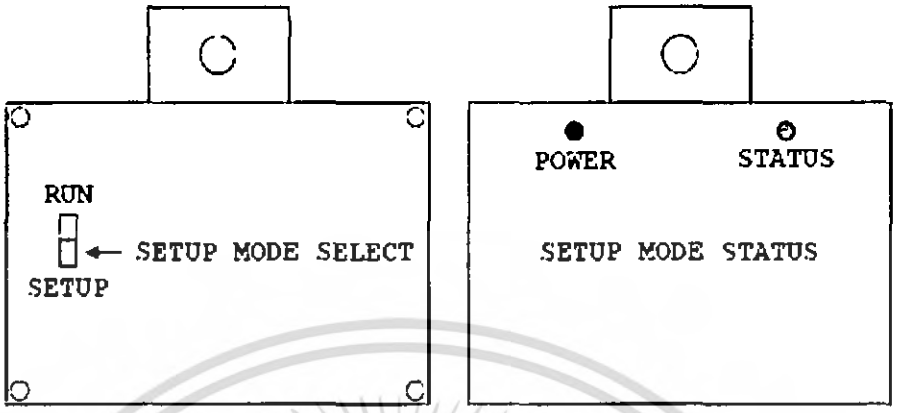
สำหรับการใช้งานเครื่อง ET-RF24G V1.0 ในโหมด *RF Auto Direction* นี้ การรับและส่งข้อมูล ด้าน RS232 จะไม่มีการตรวจสอบความพร้อมของฝ่ายรับและส่ง ด้วยสัญญาณทางไฟฟ้า (CTS\RST) เหมือนกับการใช้งานใน 2 โหมดที่ผ่านมาแล้ว โดยเมื่อมันสามารถรับข้อมูลจาก RF ได้ก็จะทำการส่งข้อมูลนั้นออกไปทางขา TX (transmit) ของ RS232 ในทันทีโดยไม่สนใจว่า อุปกรณ์ที่ต่อไว้ด้าน RS232 จะพร้อมรับข้อมูลหรือไม่ซึ่งถ้าด้าน RS232 ไม่พร้อมรับข้อมูลก็จะทำให้ข้อมูล Byte นั้นสูญหายไปทันที ซึ่งในการใช้งานนั้นผู้ใช้ควรกำหนดค่าความเร็วในการรับส่งข้อมูลด้าน RS232 ที่จะใช้กับเครื่อง ET-RF24G V1.0 ทุกๆตัวด้วยค่าความเร็วที่เท่ากัน เพื่อให้การรับและส่งข้อมูลเกิดความสัมพันธ์กันอย่างเหมาะสม

สำหรับความสามารถในการรอรับข้อมูลจาก RS232 ของเครื่อง ET-RF24G V1.0 ในโหมดนี้จะสามารถรับข้อมูลได้อย่างต่อเนื่องสูงสุด ไม่เกิน 64 Byte ดังนั้นในกรณีที่มีการส่งข้อมูลจากด้าน RS232 ด้วยข้อมูลจำนานมากกว่า 64 Byte ต่อเนื่องกันนั้น ควรทำการแบ่งข้อมูลออกเป็นชุดๆ โดยให้มีขนาดไม่เกินชุดละ 64 Byte ซึ่งหลังจากทำการส่งข้อมูลชุดถัดไปได้ 1 ชุด (64 Byte) แล้วควรทำการหน่วงเวลาไว้ขณะหนึ่งอย่างน้อย 1 mS แล้วจึงเริ่มส่งข้อมูลชุดถัดไป สลับกับการหน่วงเวลาอย่างนี้เรื่อยๆ เพื่อให้เครื่อง ET-RF24G V1.0 สามารถนำข้อมูลที่รับได้จากด้าน RS232 ส่งออกไปทางด้าน RF ได้ทันที ซึ่งถ้าทำการส่งข้อมูลอย่างต่อเนื่องโดยไม่มีการหน่วงเวลาเลยอาจทำให้ข้อมูลบาง Byte เกิดการสูญหายไปได้

2.4.2.2 การใช้งานเครื่อง ET-RF24G V1.0 ใน setup Mode

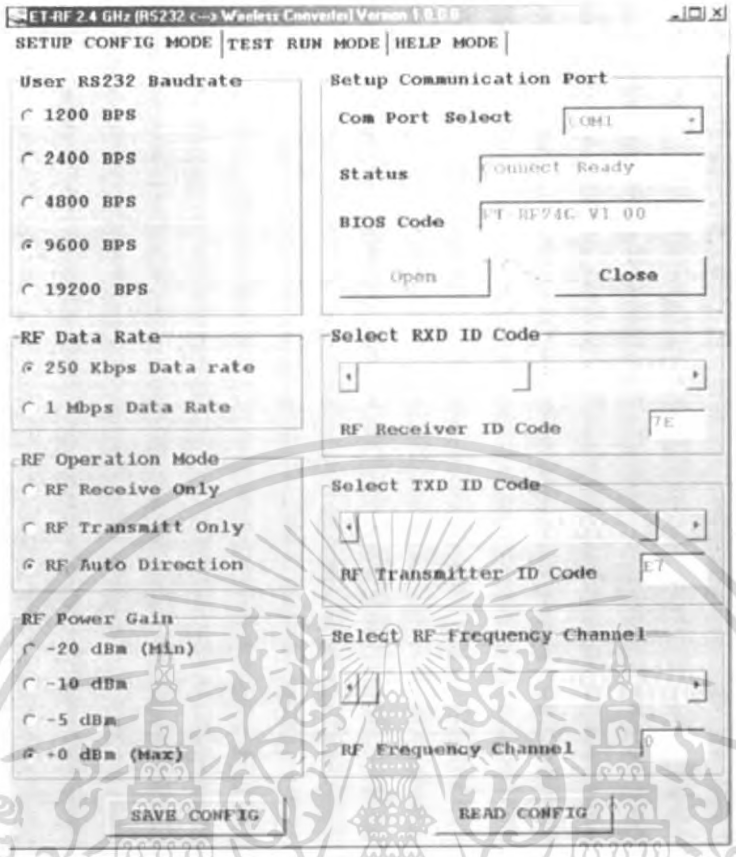
การใช้งานเครื่อง ET-RF24G V1.0 ใน setup Mode ซึ่งเป็นโหมดสำหรับใช้กำหนดค่า Configuration ต่างๆสำหรับควบคุมการทำงานของเครื่อง ET-RF24G V1.0 ที่จะใช้ในขณะที่เครื่องทำงานอยู่ใน Run Mode โดยในการ Setup ค่า Configuration ต่างๆนั้นจะกระทำร่วมกับโปรแกรม "ET_RF24G_V1.EXE" ของอีทีที ซึ่งเมื่อเครื่อง ET-RF24G V1.0 เข้าทำงานในโหมด Setup แล้ว จะสังเกตเห็นหลอดไฟแสดงสถานการณ์ทำงาน

LED STATUS จึงจะกระพริบตามจังหวะของการรับส่งข้อมูล แต่ถ้ายังไม่มีการรับส่งข้อมูลกัน LED STATUS จะติดค้างอยู่ตลอดเวลา



รูปที่ 2.20 แสดงการเลือกโหมดการทำงานสำหรับกำหนดค่า Configuration (Setup Mode)

ซึ่งการกำหนดค่า Configuration ให้กับ ET-RF24G V1 0 นั้น จะต้องกระทำในขณะที่ตัวเครื่องทำงานอยู่ใน Setup Mode เท่านั้น (เลือก Switch กำหนดโหมดไว้ทางด้าน Setup แล้วจ่ายไฟให้เครื่องเริ่มต้นทำงาน) โดยค่าของ Configuration ต่างๆ นั้นจะถูกใช้สำหรับเป็นเงื่อนไขในการทำงานของ ET-RF24G V1.0 ในขณะที่อยู่ใน Run Mode ดังนั้น ก่อนการเริ่มต้นใช้งานเครื่องในครั้งแรก จึงจำเป็นอย่างยิ่งที่จะต้องทำการกำหนดค่าของ Configuration ต่างๆให้ถูกต้องและตรงกับความต้องการที่จะใช้งานเสียก่อน โดยเมื่อทำการกำหนดค่าตัวเลือกต่างๆของ Configuration เรียบร้อยแล้ว ก็สามารถเปลี่ยนโหมดการทำงานของตัวเครื่องกลับเป็น Run Mode พร้อมกับการปิดไฟที่จ่ายให้กับตัวเครื่อง (Power-OFF) ชั่วขณะหนึ่ง จากนั้นจึงเริ่มต้นจ่ายไฟให้กับตัวใหม่ (Power-ON) ก็สามารถใช้งาน ET-RF24G V1.0 ตามค่าของ Configuration ที่กำหนดไว้แล้วได้ทันที โดยค่าเลือกต่างๆของ Configuration ที่ได้กำหนดไว้แล้วจะถูกเก็บไว้ภายในตัวเครื่องอย่างถาวร ถึงแม้ว่าจะไม่ได้ทำการจ่ายไฟให้กับตัวเครื่องแล้วก็ตาม ดังนั้นเมื่อทำการกำหนด Configuration ต่างๆเรียบร้อย ถ้าไม่มีการเปลี่ยนเงื่อนไขการทำงานของตัวเครื่องต่างไปจากเงื่อนไขเดิมที่ได้กำหนด ก็ไม่จำเป็นต้องทำการกำหนด Configuration ใหม่อีกแต่อย่างใด โดยทุกครั้งที่เริ่มต้นจ่ายไฟเข้าเครื่องในครั้งแรกนั้น การทำงานของ ET-RF24G V1.0 จะเป็นไปตามเงื่อนไขกำหนดไว้ใน Configuration เสมอทุกๆครั้ง โดยคุณสมบัติของ Configuration ต่างๆนั้นมีดังนี้



รูปที่ 2.21 แสดงโปรแกรมที่ใช้สำหรับกำหนดค่า Configuration ของ ET-RF24G V1.0

- **User RS232 Baudrate** ใช้สำหรับกำหนดค่าความเร็วในการรับส่งข้อมูลทางด้าน RS232 ของตัวเครื่องในขณะที่ทำงานอยู่ใน Run Mode ซึ่งสามารถกำหนดได้ 5 ค่าคือ
  - 1200 BPS
  - 2400 BPS
  - 4800 BPS
  - 9600 BPS
  - 19200 BPS

- **RF Data Rate** ใช้สำหรับกำหนดความเร็วในการรับส่งข้อมูลด้าน RF ของ ET-RF24G V1.0 ซึ่งจะต้องกำหนดให้เครื่อง ET-RF24G V1.0 ทุกๆตัว ที่จะนำมาใช้ติดต่อสื่อสารกัน มีค่าอัตราความเร็วในการรับส่งข้อมูลด้าน RF หรือ RF Date Rate นี้มีค่าเท่ากันทั้งหมด ซึ่งถ้ากำหนดค่าความเร็วต่างกันจะไม่

สามารถรับส่งข้อมูลกันได้ ซึ่งค่าอัตราความเร็วในการส่งข้อมูลนี้จะมีผลต่อระยะทางการรับส่งข้อมูลด้วย ซึ่งถ้าใช้ความเร็วในการส่งสูง (1 Mbps) จะทำให้รัศมีการรับส่งข้อมูลได้ระยะทางสั้นลง แต่ถ้าใช้ความเร็วในการรับส่งข้อมูลที่ช้าลง (2500 Kbps) จะทำให้ได้รัศมีการรับส่งไกลขึ้น โดยค่า RF Date Rate สามารถกำหนดได้ค่า 2 ค่า คือ

- 250 Kbps
- 1 Mbps

- **RF Operation Mode** ใช้สำหรับกำหนดโหมดการทำงานของ ET-RF24G V1.0 ซึ่งสามารถกำหนดหน้าที่การทำงานได้ 3 แบบด้วยกันคือ
  - RF Receive Only เป็นการกำหนดให้ ET-RF24G V1.0 ทำหน้าที่เป็นฝ่ายรอรับข้อมูลทางด้าน RF เพื่อเปลี่ยนเป็นข้อมูลแบบ RS232 และส่งออกไปทางด้านขา TX ของ RS232 ตลอดเวลา
  - RF Transmit Only เป็นการกำหนดให้ ET-RF24G V1.0 ทำหน้าที่เป็นฝ่ายรอรับข้อมูลทางด้าน RS232 จากขา RX เพื่อเปลี่ยนเป็นข้อมูลแบบ GFSK และส่งออกไปทางด้าน RF ตลอดเวลา
  - RF Auto Direction เป็นการกำหนดโหมดการทำงานแบบ Half Duplex 2 ทิศทาง ซึ่งสามารถสลับโหมดการทำงานระหว่างการรับและส่งข้อมูลได้เองโดยอัตโนมัติ โดยในโหมดการทำงานนี้เครื่อง ET-RF24G V1.0 จะรอตรวจสอบข้อมูลทั้งจากทางด้าน RS232 และด้าน RF จากนั้นก็กำหนดทางด้าน RS232 ก็จะทำการแปลงแล้วส่งออกทางด้าน RF จากนั้นก็จะกำหนดให้ด้าน RF กลับมาเป็นฝ่ายรอรับข้อมูลตามเดิม และเมื่อได้รับข้อมูลจากด้าน RF ก็จะแปลงเป็นข้อมูลแล้วส่งออกไปทางด้าน RS232 โดยอัตโนมัติ

- **RF Frequency Channel** เป็นการกำหนดค่าของช่องความถี่ที่ใช้ในการรับส่งข้อมูลกัน โดยสามารถเลือกกำหนดช่องความถี่ได้สูงสุดมากถึง 125 ช่อง (0-124) โดยการที่เครื่อง ET-RF24G V1.0 จะทำการรับส่งข้อมูลกันได้นั้นจะต้องกำหนดช่องความถี่ RF Frequency Channel ได้นั้น จะมีประโยชน์เป็นอย่างมากในกรณีที่มีการใช้เครื่อง ET-RF24G V1.0 จำนวนหลายๆกลุ่ม ในบริเวณพื้นที่ใกล้เคียงกัน โดยให้กำหนดช่องความถี่ ET-RF24G V1.0 กลุ่มที่จะสื่อสารข้อมูลไว้ร่วมกันที่ช่องความถี่เดียวกัน ส่วนกลุ่มอื่นๆให้เลือกกำหนดช่องความถี่ที่แตกต่างกันออกไปเพื่อลดปัญหาการรบกวนกัน

ข้อแนะนำ

ในการกำหนดค่า Configuration ให้กับเครื่อง ET-RF24G V1.0 นั้น สามารถเลือกกำหนดได้ตามความต้องการและจุดประสงค์ของการใช้งานโดยแต่ละโหมดของการใช้งานนั้นจะมีค่า Configuration ที่เหมาะสมต่างกันซึ่งขอแนะนำวิธีการกำหนดค่า Configuration ดังแนวทางต่อไปนี้

- ความเร็วในการรับส่งข้อมูลด้าน RS232 หรือ User RS232 Baudrate ที่ความเร็ว 19200 Bps นั้นเหมาะกับการใช้งาน ET-RF24G V1.0 แบบ Receive Only หรือ Transmit Only ซึ่งมีการตรวจสอบความพร้อมของสัญญาณในการรับส่งข้อมูลกันด้วย แต่ถ้าต้องการใช้เครื่อง ET-RF24G V1.0 ในโหมด Auto Direction นั้นควรกำหนดค่า User RS232 Baudrate ไว้ที่ความเร็วไม่เกิน 9600 Bps จะดีที่สุด และควรกำหนดค่า Baudrate ของทั้งสองฝ่ายให้มีค่าเท่ากันด้วย
- ค่าความเร็วของการรับส่งข้อมูลด้าน RF หรือ RF Date Rate ที่สามารถรับส่งข้อมูลกันได้ระยะทางไกลมากที่สุดและมีโอกาสผิดพลาดน้อยที่สุด คือ 250 Kbps
- ค่า RF Power Gain ที่ดีที่สุดคือ 0 dBm ซึ่งเป็นค่ากำลังสูงสุด ซึ่งจะทำให้สามารถส่งข้อมูลได้ระยะทางไกลที่สุด แต่ถ้าระยะการรับส่งข้อมูลไม่ไกลมากและมีการใช้งานเครื่อง ET-RF24G V1.0 จำนวนหลายๆกลุ่มในพื้นที่ใกล้เคียงก็อาจทำการลดกำลังส่งให้ต่ำสุดเพื่อลดปัญหาการรบกวนกันหรือกำหนดช่องความถี่ RF Frequency Channel ให้ห่างกันมากๆ
- ในกรณีที่มีการใช้เครื่อง ET-RF24G V1.0 หลายๆกลุ่มในพื้นที่ใกล้เคียงกัน ควรกำหนดช่องความถี่ในการใช้งาน หรือ RF Frequency Channel ให้ห่างกันด้วยเพื่อป้องกันการรบกวนกัน
- การใช้เครื่อง ET-RF24G V1.0 แบบ Auto Direction นั้น ถ้ามีการส่งข้อมูลจำนวนมากๆควรจัดแบ่งข้อมูลออกเป็นชุดๆ โดยให้มีขนาดข้อมูลชุดละไม่เกิน 64 Byte โดยในการส่งข้อมูลแต่ละชุดนั้นให้ทำการส่งข้อมูลอย่างต่อเนื่องโดยให้ข้อมูลแต่ละ Byte มีระยะเวลาห่างกันไม่เกิน 2.5 mS เนื่องจากถ้าข้อมูลขาดหายไปนานกว่านี้ เครื่อง ET-RF24G V1.0 จะทำการเปลี่ยนโหมดของการส่งข้อมูลกลับเป็นโหมดของการรับข้อมูลแทน ซึ่งเมื่อมีการส่งข้อมูล Byte ถัดไปมาอีกก็จะต้องเสียเวลาในการสลับโหมดจากฝ่ายรอรับข้อมูลให้เป็นฝ่ายส่งข้อมูลอีก ซึ่งจะทำให้ประสิทธิภาพในการจัดส่งข้อมูลลดลง เนื่องจากต้องเสียเวลาในการสลับโหมดการทำงานของวงจรภาค RF อยู่ตลอดเวลา โดยที่เมื่อทำการจัดส่งข้อมูลครบ 64 Byte ให้ทการหน่วงเวลาไว้ชั่วขณะหนึ่ง ประมาณ 1 mS - 2 mS แล้วจึงส่งข้อมูลชุดถัดไปอีกอย่างนี้เรื่อยๆ จะทำให้การรับส่งข้อมูลมีประสิทธิภาพสูงสุด
- การใช้งานเครื่อง ET-RF24G V1.0 แบบ Auto Direction นั้น ควรหน่วงเวลาในการสลับโหมด จากฝ่ายของการรอรับข้อมูลเป็นฝ่ายส่งข้อมูลอย่างน้อยที่สุด 3 mS – 5 mS ซึ่งถ้าส่งข้อมูลย้อนกลับด้วยเวลาที่เร็วกว่านี้อาจทำให้ฝ่ายตรงข้ามไม่สามารถรับข้อมูล Byte แรกได้ทัน

บทที่ 3

การประยุกต์ใช้ไมโครคอนโทรลเลอร์ AVR ด้วยภาษาซี

การเขียนโปรแกรมสำหรับไมโครคอนโทรลเลอร์ด้วยภาษาแอสเซมบลีนั้นมีโครงสร้างของโปรแกรมจะดูยาก การควบคุมทิศทางของโปรแกรมก็เข้าใจได้ยาก แม้แต่ผู้ที่เขียนโปรแกรมเองเมื่อเวลาผ่านไปแล้วกลับมาดูโปรแกรมใหม่ก็อาจดูงานที่ตัวเองเขียนได้ไม่เข้าใจ แม้ว่าจะเขียนคำอธิบายไว้ทุกบรรทัดก็ตาม ในอดีตการเขียนโปรแกรมด้วยภาษรซีสำหรับไมโครคอนโทรลเลอร์นั้นเมื่อแปลออกมาเป็นรหัสของภาษาเครื่องจะได้รหัสที่มีขนาดใหญ่กว่าการเขียนด้วยภาษาแอสเซมบลีมาก แต่ปัจจุบันคอมไพเลอร์ส่วนใหญ่จะแปลภาษาออกมาได้รหัสที่มีขนาดเล็กเกือบใกล้เคียงกับภาษาแอสเซมบลี การเขียนโปรแกรมด้วยภาษาซีสามารถนำไปใช้กับงานที่ซับซ้อนได้ เช่น งานควบคุมแบบป้อนกลับ (Close Loop Control) การใช้ไมโครคอนโทรลเลอร์ควบคุมเครื่องมือวัดต่างๆ (Microcontroller Based Bnstrumentation) และการใช้ไมโครคอนโทรลเลอร์ในการควบคุมหุ่นยนต์ เป็นต้น

3.1 โครงสร้างภาษาซี

ด้วยภาษาซีเป็นภาษาที่สามารถเขียนโปรแกรมเป็นแบบโครงสร้างได้ โดยโปรแกรมจะแบ่งการทำงานต่างๆ ออกเป็นกลุ่ม ๆ หรือ ฟังก์ชัน โดยฟังก์ชันเหล่านั้นสามารถเรียกขึ้นมาใช้ใหม่ได้ ในการเขียนโปรแกรมจะต้องระบุไว้ว่าในโปรแกรมนั้นมีฟังก์ชันใดให้ใช้บ้าง แต่ทุกโปรแกรมจะต้องมีฟังก์ชันหลักที่ชื่อว่า main() เสมอ พิจารณาตัวอย่างโครงสร้างของโปรแกรมต่อไปนี้

```
#include<mega128.h>          /*Preprocessor*/
#include<stdio.h>
void func1 (void);           /*Por to type*/
int func2 (int x);
void main ()                 /* ฟังก์ชันหลัก*/
{       int a;               /*ประกาศตัวแปร*/
        PORTA = 0xFF;
        func1();             /*เรียกใช้ฟังก์ชัน*/
        a = func2 (4),       /*เรียกใช้ฟังก์ชันที่มีการส่งค่า*/
        PORTA = a,
}
```

```
void func1(void)                    /*ฟังก์ชันที่ไม่มีการส่งค่า*/
{
        **************
        **************
}
int func2(intx)                     /*ฟังก์ชันที่มีการส่งค่า*/

{
        return(x*2);
}
```

จากโปรแกรมพบว่า ส่วนประกาศโพรโตไทป์จะบอกว่าโปรแกรมนี้มีฟังก์ชันชื่อ func1 ให้ใช้งาน และฟังก์ชันนี้จะทำงานเป็นโปรแกรมย่อยเพราะมี void นำหน้า และมีฟังก์ชันชื่อ func2(int x) ซึ่งรับค่าเข้าไปผ่านทางตัวแปร x และคืนค่าออกมาเป็นจำนวนเต็ม

3.2 ตัวแปรและค่าคงที่

การใช้งานตัวแปรและค่าคงที่ต่างๆ จะต้องมีการประกาศชื่อตัวแปรขึ้นมาเสียก่อนเมื่อมีการคอมไพล์โปรแกรมตัวคอมไพเลอร์จะเตรียมพื้นที่ในหน่วยความจำแรมเอาไว้สำหรับเก็บตัวแปรและค่าคงที่เหล่านั้นในการประกาศตัวแปรสามารถทำได้ดังนี้

ประเภทของข้อมูล ชื่อตัวแปร[,..... .,];

การประกาศตัวแปรจะต้องเริ่มด้วยชื่อประเภทของข้อมูล และตามด้วยชื่อตัวแปร โดยจะประกาศกี่ตัวก็ได้ ส่วนชื่อของตัวแปรนั้นจะซ้ำกันไม่ได้และต้องไม่ซ้ำกับชื่อของคำสงวน (Keywords) ของคอมไพเลอร์ตัวนั้นๆ สำหรับประเภทของข้อมูลในการประกาศตัวแปรดังนี้

ประเภทของข้อมูล	ขนาด (บิต)	ค่าที่เก็บได้
bit	1	0 ถึง 1
char	8	-128 ถึง 127
unsigned char	8	0 ถึง 255
int	16	-32768 ถึง 32767
unsigned int	16	0 ถึง 65535
long	32	-2147483648 ถึง -2147483647
unsigned long	32	0 ถึง 4294967295
float	32	-1.17549e-38ถึง 3.402823 e+38

ตารางที่ 3-1 แสดงการประกาศตัวแปรต่างๆ

สำหรับคำสงวนเป็นคำที่คอมไพเลอร์รู้จักและจะถูกใช้งานเฉพาะ เราไม่สามารถนำมาตั้งเป็นชื่อตัวแปรและฟังก์ชันได้ คำสงวนของโปรแกรมเป็นดังต่อไปนี้

_at_	idata	sfr
alien	interrupt	sfr16
bdata	large	small
bit	pdata	_task_
code	_priority_	using
compact	reentrant	Xdata
data	sbit	

3.3 ตัวดำเนินการในภาษาซี

ตัวดำเนินการจะเป็นตัวที่ใช้กระทำกับตัวแปร ค่าคงที่ต่างๆ ให้รวมเป็นค่าเดียวกันโดยอาจกระทำทางคณิตศาสตร์หรือกระทำทางลอจิกก็ได้ ในการเขียนโปรแกรมด้วยภาษีนั้น ตัวดำเนินการจะแบ่งออกเป็นสองกลุ่มใหญ่ๆ คือตัวดำเนินการที่กระทำกับตัวถูกกระทำตัวเดียว (Single Operand Operators) และตัวดำเนินการที่กระทำกับตัวถูกกระทำสองตัว (Two Operands Operators)

3.3.1 ตัวดำเนินการที่กระทำกับตัวถูกกระทำตัวเดียว

ตัวดำเนินการประเภทนี้จะกระทำกับตัวถูกกระทำเพียงตัวเดียว ประกอบด้วยตัวดำเนินการต่างๆ ดังนี้

- ลบ (negate)

~ กลับค่าลอจิกของบิตข้อมูล (bit wise complement)

! กลับค่าทางลอจิก (logical complement)

++ เพิ่มค่าขึ้นหนึ่งค่า (increment)

-- ลดค่าลงหนึ่งค่า (decrement)

* ตัวดำเนินการทางพอยน์เตอร์

& ตำแหน่งหน่วยความจำของตัวแปร

3.3.2 ตัวดำเนินการที่กระทำกับตัวถูกกระทำของตัว

ตัวดำเนินการแระเภทนี้จะกระทำกับตัวถูกกระทำสองตัว ถ้าหากมีตัวถูกกระทำหลายๆ ตัวสามารถนำมาเขียนรวมกันเป็นประโยคได้ ซึ่งประกอบไปด้วยตัวดำเนินการที่ใช้กำหนดค่า ตัวดำเนินการทดสอบค่าซึ่งจะให้ผลลัพธ์เป็นค่าทางลอจิก (จริง, เท็จ) ตัวดำเนินการทางคณิตศาสตร์ และตัวดำเนินการทางลอจิกดังต่อไปนี้

= กำหนดค่าในประโยค (assignment)

+ บวก

- ลบ

* คูณ

/ หาร (division)

% หารแบบบวก (modulo)

&& การแอนด์ (logical AND)

|| การออร์ (logical OR)

& การแอนด์แบบบิตต่อบิต (bit wise AND)

| การออร์แบบบิตต่อบิต (bit wise OR)

^ การเอ็กคลูซีฟออร์แบบบิตต่อบิต (bit wise exclusive OR)

<< เลื่อนบิตไปทางซ้าย

>> เลื่อนบิตไปทางขวา

= = ทดสอบว่าเท่ากันหรือไม่

!= ทดสอบว่าไม่เท่ากันหรือไม่

> ทดสอบว่ามากกว่าหรือไม่

< ทดสอบว่าน้อยกว่าหรือไม่

>= ทดสอบว่ามากกว่าหรือเท่ากันหรือไม่

<= ทดสอบว่าน้อยกว่าหรือเท่ากันหรือไม่

3.4 ประโยคควบคุมในภาษาซี

การทำงานของโปรแกรมนั้นจะทำคำสั่งแต่ละคำสั่งเรียงลำดับกันไป และเราสามารถให้โปรแกรมตัดสินใจในการเลือกทำได้ หรือให้ทำงานใดงานหนึ่งซ้ำๆ ตามเงื่อนไขที่กำหนดได้โดยใช้คำสั่งควบคุมในภาษาซีนั้นจะมีประโยคคำสั่งควบคุมที่ใช้ในการเลือกทำและทำงานซ้ำๆ ดังนี้

3.4.1 ประโยค IF/ELSE

```
if(expression)
        statement,
```

ถ้าหากเป็นการทำงานเลือกทำแบบมีสองทางเลือก และต้องการทำงานเพียงอย่างใดอย่างหนึ่งจะใช้ประโยค if-else ซึ่งมีรูปแบบดังนี้

```
if (expression)
        statement 1;
else
        Statement 2,
```

3.4.2 ประโยค switch

การเลือกทำที่มีทางเลือกหลายๆ ทางเลือกนั้นเราสามารถนำประโยค if-else มาซ้อนกันได้ แต่ทำให้มองดูเข้าใจยาก ในภาษาซีจะมีประโยค switch ที่ใช้ในการเลือกทำอย่างใดอย่างหนึ่งจากหลายๆทางเลือกโดยมีรูปแบบของประโยคดังนี้

```
Switch(k)
{
        case 1:  statement 1;
                 break;
        case2:   statement 2;
                 Break,
        case3:   statement 3,
                 break;
```

```
        case4:   statement 4,
                 break,

                 : :

                 : :
        default: statement n;
    }
```

3.5 การทำซ้ำ

คำสั่งให้โปรแกรมทำงานซ้ำถือว่าเป็นประโยคคำสั่งควบคุมอย่างหนึ่ง การทำซ้ำหรือที่เรียกว่าการทำลูปนั้นจะมีประโยคคำสั่งอยู่ 3 ประเภทคือ for, while และ do-while ซึ่งแต่ละแบบจะต่างกันตรงเงื่อนไขของการทำซ้ำ

3.5.1 ประโยค for

ประโยคคำสั่งนี้จะใช้ในกรณีที่มีคำจำนวนรอบของการทำซ้ำที่แน่นอน โดยมีรูปแบบดังนี้

```
for (initialization : condition : increment)
        statement ;
```

โดยที่ initialization เป็นค่ากำหนดเริ่มต้นให้กับตัวแปรของการทำลูป condition เป็นเงื่อนไขที่ใช้ทดสอบการทำซ้ำครั้งต่อไป ซึ่งจะเป็นการกระทำลอจิก increment เป็นการเพิ่มค่าให้ตัวแปรในการทำซ้ำแต่ละครั้ง สำหรับ statement จะกเป็นสเตตเมนต์ของคำสั่งที่จะทำซ้ำ ซึ่งอาจเป็นสเตตเมนต์รวมก็ได้

3.5.2 ประโยค while

การทำซ้ำแบบนี้จะตรวจสอบเงื่อนไขก่อนการทำซ้ำ ถ้าเงื่อนไขเป็นจริงจะทำสเตตเมนต์ที่กำหนดและทดสอบเงื่อนไขใหม่ ถ้าเงื่อนไขเป็นเท็จจะออกจากการทำซ้ำทันที โดยมีรูปแบบดังนี้

```
while (expression)
        statement 1;
```

3.5.3 ประโยค do – while

การทำซ้ำประเภทนี้จะตรวจสอบเงื่อนไขภายหลังการทำสเตตเมนต์แต่ละครั้ง ถ้าหากเงื่อนเป็นเท็จจะออกจากการทำซ้ำทันทีโดยมีรูปแบบดังนี้

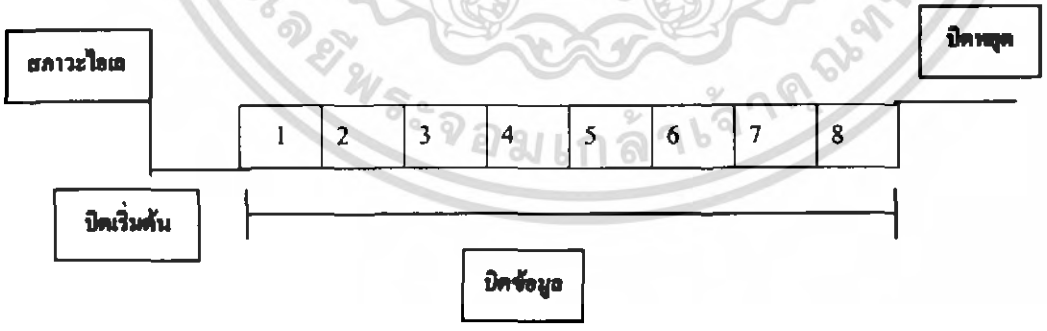
```
do {
        statement ,
}
while (condition);
```

3.6 Microcontroller เขียนโปรแกรมควบคุม AVR ผ่าน Serial Port

โดยปกติแล้ว AVR จะเป็นไมโครคอนโทรเลอร์ที่มีความสามารถในการ รับข้อมูลจากภายนอกและนำมาประมวลผล พร้อมทั้งสามารถส่งสัญญาณ เพื่อทำการควบคุมการทำงานของอุปกรณ์ต่างๆ ได้อย่างดี และในส่วนของการติดต่อสื่อสารข้อมูล (Data Communication) กับระบบภายนอกอื่นๆ ก็สามารถกระทำ โดยผ่านทางพอร์ทอนุกรม (Serial Port) ซึ่งพอร์ทอนุกรมนี้ จะเป็นส่วนที่เหมาะสมในการรับ หรือส่งข้อมูลในระยะทางไกลได้ดีกว่าพอร์ทขนาน

3.6.1 การส่งข้อมูลแบบอนุกรมในโหมด 1 (Standard UART)

การส่งข้อมูลในโหมดนี้จะเป็นการส่งข้อมูลแบบอะซิงโครนัส 8 บิตข้อมูล 1 บิตเริ่มต้นและ 1 บิตหยุดดังรูปที่ 3-1 การทำงานในโหมดนี้จะทำโดยการกำหนดข้อมูลในรีจีสเตอร์ SCON บิต SM0 และบิต SM1 ให้มีค่าเป็น "01" ซึ่ง เป็นการกำหนดให้รีจีสเตอร์ SBUF กลายเป็นตัวรับส่งข้อมูลขนาด 10 บิตแบบฟูลดูเพลกซ์ (Full Duplex) ซึ่งสามารถรับ และส่งข้อมูลได้ภายในเวลาเดียวกัน โดยใช้ขา RxD ทำหน้าที่รับสัญญาณอนุกรมที่เข้ามา และขา TxD ทำการส่งข้อมูล แบบอนุกรมไปภายนอก



รูปที่ 3-1 แสดงการส่งข้อมูลของพอร์ตอนุกรม

การส่งข้อมูลจะเริ่มต้นด้วยการส่งบิตเริ่มต้น (Start bit) ออกไปแล้วตามด้วยบิตข้อมูล (โดยส่งบิต 0 ออกไปก่อน) จากนั้นจึงเป็นการส่งบิตหยุด (Stop bit) แฟล็ก TI จะเซ็ตเมื่อส่งข้อมูลครบทั้ง 10 บิต

การรับข้อมูลจะเริ่มจากลอจิกในสายสัญญาณเปลี่ยนสภาวะจาก 1 เป็น 0 (ขอบบวกของบิตเริ่มต้น) บิตเริ่มต้น จะถูกข้ามไปไม่สนใจ จะสุ่มเอาข้อมูลอีก 8 บิตที่เหลือเขารีจีสเตอร์เลื่อนบิตภายในพอร์ทอนุกรม เมื่อครบทั้ง 8 บิตแล้ว สิ่งต่อไปนี้จะเกิดขึ้น

1. บิต 9 (บิตหยุด) จะถูกเก็บเข้าไปในบิต RB8 ของ SCON
2. SBUF จะทำการโหลดข้อมูลทั้ง 8 บิตเข้าตัวเอง
3. แฟล็ก RI เซ็ต

สิ่งที่กล่าวมาทั้งหมดจะเป็นจริงต่อเมื่อ

1. RI = 0
2. SM2 = 1 และบิตหยุดที่รับเข้ามาเป็น 1 หรืออีกกรณีหนึ่งคือ SM2 = 0

บทที่ 4

โปรแกรมวิชวลเบสิค

วิชวลเบสิก (Visual Basic) เป็นโปรแกรมที่ใช้สร้างโปรแกรมประยุกต์ สำหรับระบบปฏิบัติการวินโดว์ (Windows) เนื่องจากความง่ายของภาษาที่ใช้ในการเขียนและวิชวลเบสิกเป็นโปรแกรมที่เน้นในการสร้างแอพพลิเคชั่นเพื่อติดต่อกับผู้ใช้งานเป็นหลักทำให้ได้ผลงานออกมารวดเร็วสวยงาม และนอกจากนี้วิชวลเบสิกยังมีเครื่องมือที่ใช้ในการติดต่อสื่อสารกับพอร์ตอนุกรมซึ่งนำไปใช้ในการควบคุมเครื่องอีกด้วย

4.1 ขั้นตอนการสร้างโปรแกรมประยุกต์

การสร้างโปรแกรมประยุกต์ วิชวลเบสิก ประกอบด้วยขั้นตอนหลัก 3 ขั้นตอน คือ

1. การสร้างอินเตอร์เฟส (Interface) โดยมีฟอร์ม (Form) เป็นอ๊อบเจค (Object) พื้นฐานและเป็นที่วางตัวคอนโทรล (Control) สำหรับการติดต่อกับผู้ใช้
2. ตั้งค่าคุณสมบัติ เป็นการกำหนดพฤติกรรมและการทำงานให้กับอ๊อบเจคต่างๆ
3. การเขียนคำสั่ง เป็นการควบคุมการประมวลผลผ่านโพรซีเจอร์ (Procedure) ที่กำหนด

4.2 คอนโทรลพื้นฐาน

กลุ่มคอนโทรลพื้นฐานของวิชวลเบสิกประกอบด้วย Text Box, Label, Command Button, Picture Box, Image, Check Box, Frame เป็นต้น

ไอคอน	ชื่อตัว Control	ชื่อ Class	คำอธิบาย
☑	Check box	CheckBox	ใช้กับการเลือกแบบ ถูก/ผิด(True/False, Yes/No)
	Combo box	ComboBox	เป็นตัว Control เป็นการผสมระหว่าง Text box กับ List box ซึ่งจะปรากฏรายการ เมื่อมีการคลิกลูกศร และ Combo box ไม่สนับสนุนการเลือกแบบหลายค่า
	Command button	CommandButton	ปุ่มคำสั่งเป็นตัว Control ที่ใช้ในทุกฟอร์ม ตามปกติจะเขียนคำสั่งใน Click Event Procedure ของตัว Control นี้

ไอคอน	ชื่อตัว Control	ชื่อ Class	คำอธิบาย
	Data	Data	เป็นตัว Control ที่สามารถรวมข้อมูลกับฐานข้อมูลได้ และเป็นส่วนที่ Visual Basic ให้ผู้ใช้สามารถติดต่อระหว่างตัว Control บนฟอร์มกับฟิลด์ใน Table ของฐานข้อมูล โดย Data จะทำงานกับ Database Jet ของฐานข้อมูล แต่ไม่สามารถทำงานกับ ActiveX Data Object (ADO) ได้
	Directory List box	DirListBox	เป็น List box แบบหนึ่ง ที่แสดงไดเรคทอรีและพาร์ทที่เลือก
	Drive List box	DriveListBox	คล้ายกับ Combo box ที่ใช้เลือกชื่อของไดร์ฟในระบบ
	File list box	FileListBox	เป็น List box ชนิดพิเศษที่ใช้แสดงชื่อไฟล์ในไดเรคทอรี
	Frame	Frame	สามารถใช้เป็น Container สำหรับตัว control อื่น
	Horizontal และ Vertical Scroll Bar	HScrollBar และ VScrollBar	ใช้เป็นแถบเลื่อนแบบ Stand-alone แต่มักจะไม่ค่อยมีการใช้ เพราะตัว Control อื่น ๆ ส่วนใหญ่ จะมีแถบเลื่อนของตัวเอง แถบเลื่อนแบบ Stand-alone อาจจะใช้ในลักษณะ Slider ได้
	Image	Image	เป็นตัว Control ใช้เก็บภาพคล้ายกับ Picture Box แต่ไม่สามารถทำงานแบบ Container ได้ Image มีข้อดีที่ใช้ทรัพยากรของระบบน้อยกว่า Picture Box
	Label	Label	เป็นตัว Control ที่ใช้แสดงข้อความ หรือป้ายชื่อ
	Line	Line	เป็นตัว Control ใช้สำหรับการตกแต่งด้านกราฟฟิก
	List box	ListBox	เป็นตัว Control ที่เก็บรายการของค่า และให้ผู้ใช้เลือก ซึ่งสามารถเป็นการเลือกค่าเดียวหรือหลายค่า ขึ้นกับการกำหนดคุณสมบัติ MultiSelect
	OLE container	OLE	เป็นตัว Control ที่สามารถเป็น Host Window ให้กับโปรแกรมภายนอก เช่น Microsoft Excel หรืออาจจะกล่าวว่าเป็นการสร้าง Window ให้กับโปรแกรมอื่นบนโปรแกรมประยุกต์ Visual Basic

ไอคอน	ชื่อตัว Control	ชื่อ Class	คำอธิบาย
	Option button	OptionButton	เป็นตัว Control ใช้กับกลุ่มตัว Control โดยให้เลือกได้เพียงตัว Control เดียวต่อครั้งหนึ่ง เมื่อมีการเลือกตัว Control ในกลุ่มแล้ว ตัว Control อื่นในกลุ่มจะเปลี่ยนจากการเลือกโดยอัตโนมัติ ถ้ามีการใช้ Option Button มากกว่า 2 กลุ่ม ต้องวางแต่ละกลุ่มใน Container เช่น Frame
	Picture box	PictureBox	ใช้แสดงภาพในฟอร์แมต BMP, DIB (Bitmap), ไอคอน (Icon), WMF (Metafile), GIF และ JPEG เป็นต้น และสามารถใช้เป็น container สำหรับตัว Control อื่น
	Shape	Shape	เป็นตัว Control ใช้สำหรับการตกแต่งด้านกราฟฟิก
	Text box	TextBox	เป็นตัว Control ที่เป็นฟิลด์ ใช้เก็บตัวอักษรที่สามารถแก้ไขโดยผู้ใช้ได้ และได้รับการใช้งานมาก
	Timer	Timer	เป็นตัว Control พิเศษที่ไม่เห็นเมื่อเวลาเรียกใช้ วัตถุประสงค์การใช้คือการสร้าง Event ในฟอร์มแม่ โดยการเขียนคำสั่งใน Procedure ที่เจาะจงสำหรับการทำงานเบื้องหลัง เช่น การตรวจสถานะของอุปกรณ์ต่อพ่วง

ตารางที่ 4-1 แสดงถึงคอนโทรลพื้นฐานของวิชวลเบสิกที่ใช้ในการสร้างโปรแกรม

4.3 คุณสมบัติร่วม

1. Left, Top, Width, Height ใช้จัดตำแหน่งของอ๊อบเจค เช่น การวางฟอร์มที่มุมซ้ายบน มีความกว้าง 5000 Twips และสูง 3000 Twips

```
Form1.Left = 0
Form1.Top = 0
Form1.Width = 5000
Form1.Height = 3000
```

2. ForeColor และ BackColor ใช้กำหนดสีของข้อความและสีพื้นหลังของอ๊อบเจค ซึ่งสามารถกำหนดสีเป็นแบบ Palette และ system เช่น

กำหนดสีแบบ Palette	Label1.ForeColor = vbHighlightText
กำหนดสีแบบ System	Label1.BackColor = &H800000D

รวมถึงการกำหนดค้วยค่าคงที่, เลขฐานสิบ และเลขฐานสิบหก เช่นตัวอย่างเป็นสีเดียวกัน

ค่าคงที่	Text1.BackColor = vbCyan
เลขฐานสิบ	Text1.BackColor = 16776960
เลขฐานสิบหก	Text1.BackColor = &HFFF00

3. Font ใช้กำหนดลักษณะการแสดงตัวอักษร การตั้งค่าคุณสมบัติ Size, Bold, Italic, Underline และ Strikethrough เช่น

```
Text1.Font.Name = "Arial"
Text1.Font.Size = 12
Text1.Font.Bold = True
```

4. Caption, Text ใช้ในการแสดงข้อความ โดยคุณสมบัติ Caption เป็นข้อความที่ปรากฏภายในตัวคอนโทรลและไม่สามารถแก้ไขได้โดยผู้ใช้เมื่อเรียกใช้โปรแกรม เช่น

```
Label1.Caption = "Title"
Text1.Text = "Hello Word"
```

5. Enabled, Locked และ Visible โดยคุณสมบัติ Visible ใช้กำหนดการมองเห็น คุณสมบัติ Enabled และ Locked กำหนดการเข้าถึงตัว Control แต่ต่างกันที่ คุณสมบัติ Enabled ถ้ากำหนดไม่ให้เข้าถึง (Enabled = False) แล้ว ตัว Control ไม่สามารถรับโฟกัสได้ ส่วนคุณสมบัติ Locked ถ้ากำหนดไม่ให้เข้าถึง (Locked = True) แล้ว ตัว Control ยังสามารถรับโฟกัสได้

6. MousePointer และ MouseIcon ใช้กำหนดไอคอนของเมาส์เมื่อเคลื่อนที่ผ่านตัวคอนโทรล เช่น

```
Text1.MousePointer = vbCrosshair
```

7. คุณสมบัติอื่นๆ ได้แก่

- คุณสมบัติ Value มีค่าตามชนิดของตัวคอนโทรล
- คุณสมบัติ Index ใช้ในการสร้าง Array ของตัวคอนโทรลและมีคุณสมบัติแบบอ่านอย่างเดียว
- คุณสมบัติ Appearance เป็นลักษณะหน้าตาของคอนโทรล ตามปกติจะกำหนดค่าคุณสมบัติเริ่มต้นเป็น 3 มิติ การกำหนดค่าทำได้ในเวลาออกแบบ และเป็นแบบอ่านอย่างเดียว เมื่อมีการเรียกใช้
- คุณสมบัติ Border Style คือคุณสมบัติเส้นขอบ มีคอนโทรลสนับสนุน ได้แก่ Text box, Label, Frame, Picture box, Image และ OLE โดยถ้าตั้งค่าเป็น 0-none จะไม่มีเส้นขอบ ถ้าตั้งค่าเป็น 1-Fixed จะมีเส้นขอบ
- คุณสมบัติ ToolTip เป็นหน้าต่างสีเหลืองขนาดเล็กแสดงคำอธิบายสั้นๆ ซึ่งปรากฏขึ้นเมื่อเมาส์เคลื่อนที่ไปอยู่เหนือตัว Control หรือไอคอน

รูปที่ 4-1 แสดงถึงการกำหนดคุณสมของคอนโทรลที่ใช้ในการสร้างโปรแกรม

4.4 ทฤษฎีร่วม

1. Move ใช้ในการย้ายอ๊อบเจค หรือตัวคอนโทรลพร้อมทั้งปรับขนาดในเวลาเดียวกัน มีไวยากรณ์ คือ [object.]Move Left [, Top, Width, Height]

2. Refresh ใช้ในการวาดตัวคอนโทรลใหม่ ตามปกติไม่ใช้เนื่องจาก Visual Basic มีการปรับค่าโดยอัตโนมัติ แต่สามารถใช้ได้ในกรณีที่ต้องการให้มีการปรับคุณสมบัติทันที เช่น

```
Dim i As Integer
For i = 1000 to 1 Step - 1
Label1.Caption = (str(i))
Label1.Refresh
Next
```

3. SetFocus ใช้ย้ายโฟกัสไปที่คอนโทรลตัวที่ระบุ เช่น

```
Text1.SetFocus
```

4.5 อีเวนต์ร่วม

1. Click และ DblClick Event โดย Click Event จะเกิดเมื่อผู้ใช้กดปุ่มเมาส์ด้านซ้ายบนตัวคอนโทรล และ DblClick Event เกิดเมื่อเป็นการปุ่มเมาส์ด้านซ้ายแบบดับเบิลคลิก

2. Change Event เป็นอีเวนต์พื้นฐานที่เกิดเมื่อข้อมูลของตัวคอนโทรล มีการเปลี่ยนแปลง

3. GotFocus และ LostFocus Event ซึ่ง GotFocus Event เกิดขึ้นเมื่อตัวคอนโทรล ได้รับการโฟกัส เช่นเมื่อเมาส์เลื่อนมาถึง และ LostFocus Event จะเกิดเมื่อการโฟกัสออกไปจากตัวคอนโทรล

4. KeyPress, KeyDown และ KeyUp Event เป็นอีเวนต์ ที่เกิดเพื่อกดปุ่มแป้นพิมพ์ ขณะที่ตัวคอนโทรลได้รับการโฟกัส โดยมีลำดับดังนี้ KeyDown (เมื่อผู้ใช้กดปุ่มแป้นพิมพ์) KeyPress (Visual Basic แปลปุ่มนั้นเป็นรหัส ANSI) และ KeyUp (เมื่อปล่อยปุ่มแป้นจากพิมพ์)

5. MouseDown, MouseUp และ MouseMove Event เป็นอีเวนต์ ที่เกิดขึ้นเมื่อมีการคลิก, ปล่อย หรือย้าย ของเมาส์บนตัว Control โดยลำดับการเกิด Click Event มีลำดับ คือ MouseDown -> MouseUp -> Click -> MouseMove ลำดับการเกิด DblClick Event มีลำดับ คือ MouseDown -> MouseUp -> Click -> MouseMove -> DblClick -> MouseUp -> MouseUp

4.6 อีเวนต์ของฟอร์ม

อีเวนต์ของฟอร์มเป็นอ็อบเจคพื้นที่ใช้ในการติดต่อกับผู้ใช้งาน ดังนี้

1. Load Event เป็นอีเวนต์ที่เกิดขึ้นต่อจาก Initialize Event ใช้ในการกำหนดค่าเริ่มต้นให้กับฟอร์มและอ็อปเจคค์

2. Resize Event เกิดขึ้นก่อนที่จะเห็นฟอร์ม โดยอีเวนต์สามารถใช้ในการปรับตัว คอนโทรลให้พอดีกับฟอร์ม นอกจาก Resize event เกิดขึ้นผู้ใช้ปรับขนาดฟอร์มเอง

3. Unload Event เป็นช่วงสุดท้ายก่อนที่โปรแกรมจะถูกปิด อีเวนต์นี้เกิดเมื่อผู้ใช้ปิดโปรแกรม ก่อนที่โปรแกรมจะถูกปิด เราสามารถแจ้งข่าวสารให้ผู้ใช้ได้ เช่น ถามถึงการบันทึกข้อมูลก่อนออกจากโปรแกรม ถ้าไม่มีการยกเลิกการ Unload ขั้นต่อไป วิชวลเบสิกจะทำการยกเลิกตัว คอนโทรล ปิดฟอร์ม และยกเลิกทรัพยากรต่าง ๆ ของวินโดว์

4.7 โพรซีเจอร์และฟังก์ชั่น

โพรซีเจอร์หรือฟังก์ชั่นเป็นส่วนที่ใช้เขียนโปรแกรมลงไปซึ่งสามารถช่วยลดความซ้ำซ้อนของโปรแกรมแต่ละส่วน สำหรับโพรซีเจอร์สามารถแบ่งได้เป็น 2 ประเภท คือ

1. โพรซีเจอร์เหตุการณ์ จะทำงานเมื่อเกิดเหตุการณ์ต่างๆขึ้นกับอ็อปเจ็ค เช่น เมื่อผู้ใช้คลิกปุ่ม "Start" ถ้าหากภายในปุ่มมีโพรซีเจอร์อยู่ ก็จะเกิดการทำงานตามคำสั่งในโพรซีเจอร์นั้นทันที

2. โพรซีเจอร์ที่กำหนดเองเป็นโพรซีเจอร์แบบทั่วไปหลังจากสร้างขึ้นแล้วเราสามารถเรียกใช้ได้ทันที เพียงแค่อ้างชื่อของโพรซีเจอร์นั้นเท่านั้น

4.8 ฟังก์ชั่น

ลักษณะของฟังก์ชั่นจะคล้ายกับโพรซีเจอร์ เพียงแต่ฟังก์ชั่นจะมีการส่งค่ากลับไปยังโปรแกรมที่เรียกฟังก์ชั่นนั้นมาใช้

4.9 การประกาศตัวแปร

การประกาศตัวแปรมีคำสั่งแตกต่างกันไปตามชนิดของการใช้งานดังนี้

- Static : ตัวแปรนี้จะใช้ได้เฉพาะภายในโพรซีเจอร์เท่านั้นและค่าในตัวแปรจะถูกเก็บไว้ตลอดเวลา แม้ว่าจะออกโพรซีเจอร์ไปแล้วก็ตาม
- Dim : มีลักษณะเป็น Static แต่สามารถประกาศตัวแปรได้ 2 ที่ คือ ในโพรซีเจอร์และ General Declarations (ส่วนแรกของโปรแกรมใช้ในการประกาศตัวแปร) หากประกาศในโพรซีเจอร์แล้ว เมื่อออกจากโพรซีเจอร์แล้ว ค่าในตัวแปรจะถูกรีเซ็ทใหม่อัตโนมัติ ดังนั้นในแต่ละโพรซีเจอร์อาจมีตัวแปรชื่อซ้ำกันได้ โดยไม่มีผลต่อกัน แต่การประกาศตัวแปรบน General Declarations จะทำให้โพรซี

เจอร์ทุกตัวบนฟอร์มนั้นสามารถเรียกใช้ได้ และหากมีการเปลี่ยนแปลงค่าในโพรซีเจอร์ใดๆแล้ว ก็จะมีผลไปกับโพรซีเจอร์อื่นๆด้วย

- Private : มีรูปแบบเช่นเดียวกันกับ Dim แต่ประกาศได้ที่ General Declarations เท่านั้น และจะใช้ได้เฉพาะบนฟอร์มนั้นเท่านั้น
- Public : มีรูปแบบเช่นเดียวกันกับ Private แต่สามารถใช้ได้ในทุกๆฟอร์มบนโปรแกรมนั้น

ประเภทข้อมูล	ประเภท	ขนาด	การเก็บข้อมูล หรือ ช่วงข้อมูล
Integer	จำนวนเต็ม	2 ไบต์	–32,768 ถึง 32,767
Long	จำนวนเต็ม	4 ไบต์	–2,147,483,648 ถึง 2,147,483,647
Boolean	จำนวนเต็ม	2 ไบต์	เก็บค่า 0 และ –1 ซึ่งแทน False หรือ True
Byte	จำนวนเต็ม	1 ไบต์	เก็บค่าในช่วง 0 ถึง 255
Single	จำนวนทศนิยม	4 ไบต์	ค่าลบ –3.402823E38 ถึง –1.401298E-45 ค่าบวก 1.401298E-45 ถึง 3.402823E38
Double	จำนวนทศนิยม	8 ไบต์	ค่าลบ –1.79769313486232E308 ถึง -4.94065645841247E-324 ค่าบวก 4.94065645841247E-324 ถึง 1.79769313486232E308
Currency	จำนวนทศนิยม (4 ตำแหน่ง)	8 ไบต์	–922,337,203,477.5808 ถึง –922,337,203,477.5807
Decimal	จำนวนทศนิยม	8 ไบต์	ค่าที่ไม่มีทศนิยม +/- 79,228,162,514,264,337,593,543,950,335 ค่าที่มีทศนิยม +/- 7.92281625142643 37593543950335 และมีทศนิยม 28 ตำแหน่ง
String	ข้อความ	-	-
Date	วันที่/เวลา	8 ไบต์	เก็บค่าระหว่าง 1 มกราคม ค.ศ. 100 ถึง 31 ธันวาคม ค.ศ. 9999 และเวลาใดๆ

ตารางที่ 4-2 แสดงถึงประเภทของข้อมูลพื้นฐานที่ใช้ในการสร้างตัวแปร

4.10 คำสั่งพื้นฐาน

คำสั่งพื้นฐานที่ใช้ในการสร้างโปรแกรม ได้แก่

- คำสั่งเงื่อนไข If... Then... Else . End If มีรูปแบบดังนี้

```
IF
        < เงื่อนไขที่ทำการเปรียบเทียบ >
Then
        < ถ้าเงื่อนไขเป็นจริง เขียนโค้ดที่นี่ >
Else
        < ถ้าเงื่อนไขไม่เป็นจริง เขียนโค้ดที่นี่ >
End If
```

- คำสั่งเงื่อนไข Select Case มีรูปแบบดังนี้

```
Select Case
        < ชื่อตัวแปรที่ต้องหานำไปเปรียบเทียบ >
Case
        < ค่าที่กำหนดสำหรับเปรียบเทียบ >
        < กรณีค่าตัวแปรเปรียบเทียบตรงกับค่าที่กำหนดไว้ จะทำคำสั่งในส่วนนี้ >
Case
        < ค่าอื่นที่ต้องการเปรียบเทียบ >
        < กรณีค่าตัวแปรเปรียบเทียบตรงกับค่าที่กำหนดไว้ จะทำคำสั่งในส่วนนี้ >
Case Else
```

< กรณีนอกเหนือจากข้างต้น จะทำคำสั่งในส่วนนี้ >

End Select

- คำสั่งวนรอบแบบ For . Next มีรูปแบบดังนี้ คือ

For

< ชื่อตัวแปรประเภทตัวเลข > = < ค่าเริ่มต้น > To < ค่าสิ้นสุด >

< โปรแกรมที่ต้องการให้ทำงานวนรอบ >

Next

< ชื่อตัวแปร >

- คำสั่งวนรอบแบบ Do Until หรือ While...Loop มีรูปแบบดังนี้ คือ

Do While/Until

< เงื่อนไขการเปรียบเทียบ >

< คำสั่งที่ต้องการให้ทำงานวนรอบ >

Loop

- คำสั่งวนรอบแบบ Do...Until หรือ While.. Loop มีรูปแบบดังนี้

Do

< คำสั่งที่ต้องการให้ทำงานวนรอบ >

Loop While/Until

< เงื่อนไขการเปรียบเทียบ >

4.11 คำสั่งเกี่ยวกับการจัดการรูปภาพ

คำสั่งเกี่ยวกับการจัดการรูปภาพ ได้แก่

- LoadPicture : เพื่อทำการโหลดรูปภาพเข้ามาใส่ใน Picture Box หรือ Image มีรูปแบบ คือ

  [ชื่อ Picture Box].Picture = LoadPicture(ที่อยู่ของรูปภาพ)

- Point : ใช้อ่านค่าสีจากจุดที่กำหนดบนรูปภาพ ซึ่งจะให้ค่าสีในระบบ Long Color มีรูปแบบ คือ

  [ชื่อ Picture Box].Point (พิกัด X, พิกัด Y)

- PSet : ใช้เพื่อวาดจุดลงไปบนรูปภาพตามจุดที่กำหนด รูปแบบ คือ

  [ชื่อ Picture Box Name].PSet (พิกัด X, พิกัด Y), รหัสสีที่ต้องการ

- PaintPicture : ใช้เพื่อคัดลอกรูปภาพจากที่หนึ่งไปอีกที่ โดยสามารถปรับขนาดได้

[ชื่อกล่องรูปภาพปลายทาง].PaintPicture (ชื่อกล่องรูปภาพต้นทาง, [ตำแหน่งเริ่มต้นของรูปต้นทางแกนX], [แกนY], [ขนาดความกว้างรูป], [ความสูงรูป], [ตำแหน่งเริ่มต้นของรูปต้นทางแกนX], [แกนY], [ขนาดความกว้างรูป], [ความสูงรูป], [รูปแบบการคัดลอกภาพ])

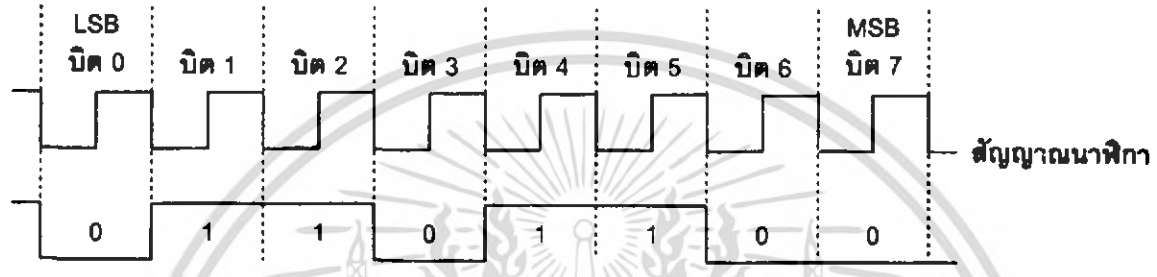
4.12 คำสั่งเกี่ยวกับการติดต่อพอร์ตอนุกรม

4.12.1 ทำความรู้จักกับพอร์ตอนุกรม

การเคลื่อนย้ายข้อมูลจากคอมพิวเตอร์ไปยังอุปกรณ์ต่อพวงอื่นๆหรือคอมพิวเตอร์ด้วยกันแบบอนุกรม เป็นการรับข้อมูลครั้งละ 1 บิต แต่ก็สามารถรับส่งข้อมูลได้คราวละหลายๆบิตได้ หากแต่จะต้องมีการตกลงกันระหว่างตัวส่งและตัวรับว่า จะรับส่งข้อมูลคราวละกี่บิต ตัวรับจะต้องรอข้อมูลมาให้ครบทุกบิตเสียก่อนจึงทำการประมวลผล ส่งผลให้การสื่อสารข้อมูลอนุกรมอาจมีความเร็วต่ำกว่าแบบขนานในด้านจำสวนสายสัญญาณ การรับส่งข้อมูลแบบอนุกรมจะใช้จำนวนสายที่น้อยกว่ามาก อย่างน้อยที่สุดใช้เพียง 2 – 3 เส้นเท่านั้น แต่อัตราเร็วในการรับส่งข้อมูลอาจต่ำกว่าแบบขนาน อย่างไรก็ตามการรับส่งข้อมูลแบบอนุกรมสามารถใช้สายสัญญาณที่มีความยาวมากกว่าแบบขนาน ทำให้ระยะทางในการสื่อสารข้อมูลแบบอนุกรมสามารถทำได้มากกว่า

4.12.2 การสื่อสารแบบอนุกรม

การสื่อสารแบบอนุกรมนั้นแบ่งออกได้เป็น 2 แบบคือการสื่อสารอนุกรมแบบซิงโครนัสและการสื่อสารอนุกรมแบบอะซิงโครนัส การสื่อสารแบบซิงโครนัสจะมีสัญญาณนาฬิการ่วมอยู่กับการรับและส่งสัญญาณด้วย ตัวอย่างการส่งข้อมูลแบบซิงโครนัสคือคีย์บอร์ดของคอมพิวเตอร์ ซึ่งสายเส้นหนึ่งจะเป็นสายของสัญญาณนาฬิกา ส่วนสายอีกเส้นจะเป็นสายของข้อมูล ดังนั้นการติดต่อกันแบบซิงโครนัสนี้จะต้องใช้สายในการเชื่อมต่ออย่างน้อยที่สุด 3 เส้นคือสัญญาณนาฬิกา,ข้อมูลและกราวด์แสดงให้เห็นถึงไทมิ่งไดอะแกรมของการส่งข้อมูลแบบซิงโครนัส



รูปที่ 4-2 รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรม

4.12.3 การสื่อสารข้อมูลแบบอะซิงโครนัส

การสื่อสารข้อมูลแบบอะซิงโครนัสคือการรับและส่งข้อมูลไปในสายโดยไม่จำเป็นต้องมีสัญญาณนาฬิการ่วมด้วยเหมือนกับการรับส่งข้อมูลแบบซิงโครนัส แต่จะใช้การกำหนดค่าสัญญาณนาฬิกาทั้งภาครับและภาคส่งให้มีค่าเท่ากัน ซึ่งเรียกสัญญาณนาฬิกาที่ใช้ในการกำหนดค่าให้ภาครับและภาคส่งนี้ว่า อัตราการถ่ายทอดข้อมูล หรือ บอดเรต (Baudrate) มีหน่วยเป็น บิตต่อวินาที (bit per second bps)

รูปแบบของข้อมูลที่ใช้ในการรับส่งแบบอะซิงโครนัสประกอบด้วย 4 ส่วนด้วยกันคือ

1. บิตเริ่มต้น (Start Bit) ซึ่งจะมีขนาด 1 บิต
2. บิตข้อมูลแบบอนุกรมจะมีขนาด 5,6,7 หรือ 8 บิต
3. บิตตรวจสอบพาริตี้ (Parity Bit) จะมีขนาด 1 บิตหรือไม่มี
4. บิตปิดท้าย (Stop Bit) จะมีขนาด 1,1.5 หรือ 2 บิต

รูปแบบของข้อมูลอนุกรมแบบอะซิงโครนัส เมื่อไม่มีข้อมูลที่จะส่ง ขา DATA จะมีสถานะลอจิก "1" ซึ่งจะเรียกสถานะนี้ว่าสถานะหยุดรอ (Waiting Stage) การเริ่มต้นส่งข้อมูลจะเริ่มจากการให้ขา DATA มีลอจิก "0" ด้วยช่วงระยะเวลา 1 บิต เรียกบิตนี้ว่า บิตเริ่มต้น จากนั้นบิตข้อมูลจะถูกส่งออกไป โดยเริ่มจากบิตที่มีนัยสำคัญต่ำสุด (LSB) ก่อน ซึ่งข้อมูลในไบต์ที่จะส่งอาจจะมีจำนวนบิต 5, 6, 7 หรือ 8 บิตก็ได้ จากนั้นตามด้วยบิตพาริตี้ ซึ่งใช้เพื่อตรวจสอบความผิดพลาดที่เกิดขึ้นจากการส่งข้อมูล บิตสุดท้ายที่ส่งคือ บิตปิดท้าย ซึ่งจะให้

ขาดาต้ามีสถานะลอจิก "1" อีกครั้งด้วยระยะเวลาอย่างน้อย 1 บิต, 1.5 บิต หรือ 2 บิต เพื่อเป็นการแสดงว่าสิ้นสุดข้อมูลแล้ว

อุปกรณ์พิเศษที่ได้รับการออกแบบมาสำหรับการรับและส่งข้อมูลแบบอะซิงโครนัสเรียกว่า Universal Asynchronous Receiver Transmitter หรือ UART อัตราความเร็วในการรับและส่งข้อมูลของการรับส่งข้อมูลแบบอะซิงโครนัสคือ ค่าบอดเรต ซึ่งก็คือค่าจำนวนบิตต่อวินาทีที่ใช้ในการรับและส่งข้อมูล บอดเรตมาตรฐานที่ใช้สำหรับพอร์ตอนุกรม RS-232 ได้แก่ 110, 150, 300, 600, 1200, 2400, 4800, 9600, และ 19200 บิตต่อวินาที และมีค่าเพิ่มมากขึ้นตามเทคโนโลยีของคอมพิวเตอร์ซึ่งการรับส่งแบบอนุกรมโดยไม่ผ่านโมเด็มอาจจะสามารถกำหนดค่าบอดเรตได้สูงถึง 11520 บิตต่อวินาที เนื่องจากบอดเรตคือจำนวนบิตของข้อมูลที่สามารถถ่ายทอดได้ภายใน 1 วินาที ยกตัวอย่าง ข้อมูลอนุกรมถูกส่งในลักษณะ 8 บิต ไม่มีการตรวจสอบพาริตี้ มีบิตเริ่มต้น 1 บิต และบิตปิดท้าย 1 บิต ความยาวของข้อมูลที่รับส่งนี้เท่ากับ 10 บิต ถ้าใช้บอดเรตในการส่งข้อมูลเท่ากับ 9600 บิตต่อวินาที ก็จะสามารถรับส่งข้อมูลได้ด้วยความเร็ว 960 ไบต์ต่อวินาที และถ้ามีการใช้พาริตี้ความเร็วในการรับส่งข้อมูลจะเหลือเป็น 872 ไบต์ต่อวินาที

รูปที่ 4.3 รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรมแบบอะซิงโครนัส

การตรวจสอบพาริตี้สามารถกำหนดให้เป็นแบบคี่ (odd), แบบคู่ (even) หรือไม่มีการตรวจสอบพาริตี้ก็ได้ การตรวจสอบพาริตี้เป็นการตรวจสอบจำนวนรวมของบิตที่เป็นลอจิก "1" ภายในข้อมูลที่ส่งไป 1 ไบต์ว่ามีจำนวนรวมเป็นเลขคู่หรือเลขคี่โดยต้องรวมบิตพาริตี้เข้าไปด้วย ยกตัวอย่าง ข้อมูลที่จะทำการส่งมีขนาด 8 บิต และมีค่าเท่ากับ 99 ฐานสิบหก หรือ 10011001 ฐานสองจะเห็นว่าข้อมูลในไบต์นี้มีจำนวนลอจิก "1" จำนวน 4 ตัวซึ่งเป็นเลขคู่ ดังนั้นถ้ากำหนดค่าพาริตี้เป็นคู่ค่าในบิตพาริตี้ จะต้องมีลอจิกเป็น "0" แต่ถ้าพาริตี้เป็นคี่ ค่าที่บิตพาริรี้จะต้องเป็น "1" เพื่อให้ข้อมูล 1 ไบต์รวมทั้งบิตพาริตี้มีจำนวนบิตที่เป็นลอจิก "1" รวมกันเป็นเลขคี่

บิตพาริตี้ถูกสร้างขึ้นจากภาคส่งข้อมูลของ UART โดยภาครับจะต้องกำหนดคุณสมบัติการตรวจสอบพาริตี้ให้ตรงกันว่า จะตรวจสอบพาริตี้คี่หรือคู่ จากนั้นภาครับของ UART จะตรวจสอบค่าพาริตี้ที่เกิดขึ้นว่าเป็นคู่หรือเป็นคี่ โดยการนับจำนวนลอจิก "1" ทั้งหมดรวมทั้งบิตพาริตี้ด้วยถ้ากำหนดพาริตี้ไว้เป็นคู่แต่อ่านค่าตัวเลขในการนับออกมาได้ตัวเลขเป็นคี่ ทางภาครับจะแจ้งข้อผิดพลาดให้ผู้ใช้ทราบ นับเป็นการตรวจสอบความผิดพลาดของการถ่ายทอดข้อมูลที่ง่ายที่สุด แต่จะเชื่อถือได้เมื่อมีบิตข้อมูลที่ทำการส่งผิดพลาดเพียงบิต

เดียวเท่านั้น ถ้าข้อมูลที่ส่งมีบิตที่ผิดพลาดมากกว่า 1 บิต การตรวจสอบด้วยวิธีนี้จะไม่ได้ผล สำหรับการตั้งพาริตี้บิตเป็น NONE นั้นทั้งภาครับและภาคส่ง จะไม่มีการตรวจสอบพาริตี้

ข้อมูล	บิตพาริตี้คู่	บิตพาริตี้คี่
00000000	0	1
00000001	1	0
00000010	1	0
00000011	0	1
00000100	1	0
11111110	1	0
11111111	0	1

ตารางที่ 4-3 *แสดงลักษณะการทำงานของบิตพาริตี้*

คอมพิวเตอร์ในรุ่น AT เกือบทั้งหมดจะใช้ UART เบอร์ 16450 และ 16550 ส่วนคอมพิวเตอร์ในรุ่น XT ใช้ UART เบอร์ 8250 UART ชิปเหล่านี้มีระดับแรงดันเป็นแบบทีทีแอล (0 และ +5V) แต่เพื่อให้มีแรงดันเป็นไปตามมาตรฐาน RS-232 และเพื่อให้การรับส่งข้อมูลสามารถทำได้ที่ระยะทางไกลมากขึ้น ระดับแรงดันทีทีแอลจะถูกแปลงเป็นระดับแรงดันที่สูงขึ้นโดยลอจิก "0" มีระดับแรงดัน +3V ถึง +12V ในขณะที่ลอจิก "1" มีระดับแรงดัน -3V ถึง -12V

4.12.4 มาตรฐานพอร์ตอนุกรมแบบ RS-232

มาตรฐานการเชื่อมต่อแบบอนุกรม RS-232 เป็นมาตรฐานอุตสาหกรรมที่ออกแบบมาเพื่อใช้ในการส่งข้อมูลอนุกรมแบบอะซิงโครนัส 2 ทิศทาง โดยมาตรฐาน RS-232 ในอดีตนั้นถูกออกแบบมาเพื่อการส่งผ่านข้อมูลจากคอมพิวเตอร์ไปยังโมเด็มเพียงอย่างเดียว เพื่อที่จะนำข้อมูลจากโมเด็มนี้สื่อสารผ่านสายโทรศัพท์ไปยังคอมพิวเตอร์อีกชุดอยู่ห่างไกลกัน โดยคณะกรรมการที่เรียกว่าสมาคมอุตสาหกรรมอิเล็กทรอนิกส์ (Electronic Industries Association : EIA) ได้วางมาตรฐานที่มีชื่อเรียกกันว่า EIA RS-232 มาตรฐานนี้ในช่วงแรกจะใช้คอนเน็กเตอร์เป็นแบบ DB-25 โดยกำหนดความสูงสุดของสายสัญญาณไว้ที่ 50 ฟุต มีระดับสัญญาณตั้งแต่ -3 ถึง -12V แสดงว่ามีข้อมูล (Mark) และ +3 ถึง +12V แสดงว่าเป็นช่องว่าง (Space)

มาตรฐาน RS-232 ได้กำหนดรูปแบบของอุปกรณ์เชื่อมต่อข้อมูล (Data Terminal Equipment . DTE) กับวงจรข้อมูลปลายทาง (Data Circuit Terminating : DCE) ไว้ว่าอุปกรณ์ DTE จะต้องเป็นอุปกรณ์ที่มีการประมวลผลในตัวเช่น ไมโครคอนโทรลเลอร์ หรือ ไมโครคอมพิวเตอร์ ซึ่งมีความสามารถในการสร้างข้อมูลแบบอนุกรมได้ ส่วนอุปกรณ์ DCE จะทำหน้าที่เป็นเพียงตัวรับข้อมูลที่ส่งมาจาก DTE เท่านั้น โดยการรับส่งข้อมูลระหว่างอุปกรณ์ทั้งสองจะกระทำผ่านมาตรฐาน RS-232

ข้อแตกต่างของอุปกรณ์ DTE และอุปกรณ์ DCE อย่างหนึ่งที่เห็นได้ชัดคือ คอนเน็กเตอร์ของ DTE จะเป็นตัวผู้ ส่วนคอนเน็กเตอร์ของ DCE จะเป็นตัวเมีย ซึ่งพอร์ตอนุกรมของคอมพิวเตอร์ที่ใช้กันอยู่ทั่วไปจะเป็นแบบ DTE ส่วนคอนเน็กเตอร์ที่อยู่ในโมเด็มจะเป็นแบบ DCE

สำหรับการใช้งานบนคอมพิวเตอร์ พอร์ตอนุกรม RS-232 มักถูกใช้เชื่อมต่อกับโมเด็มหรือเมาส์ โดยสามารถรับส่งข้อมูลได้ที่ความยาวของสายสัญญาณสูงสุดถึง 20 เมตร

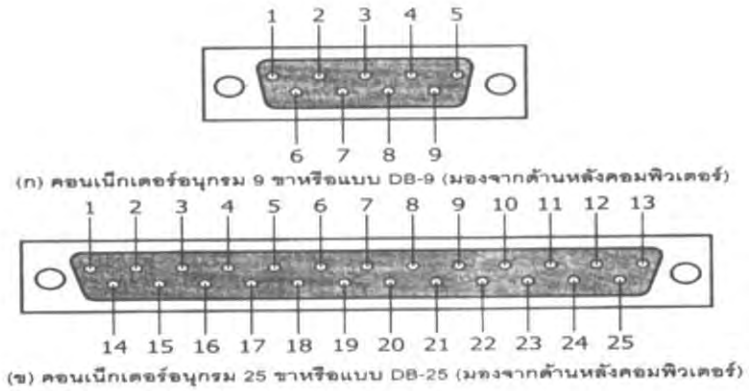
4.12.5 คอนเน็กเตอร์สำหรับพอร์ต RS-232 และการเชื่อมต่อ

มาตรฐานการเชื่อมต่อแบบ RS-232 จะใช้คอนเน็กเตอร์แบบ DB-25 ตัวผู้หรือ DB-9 ตัวผู้ซึ่งคอนเน็กเตอร์แบบ DB-25 จะมีขาต่อใช้งานเพียง 9 เส้นเช่นเดียวกับคอนเน็กเตอร์แบบ DB-9 เนื่องจากขาอื่น ๆ ที่เคยใช้งานในอดีต ปัจจุบันมีการใช้งานไม่มากนัก จึงถูกยกเลิกไป

สำหรับการเชื่อมต่อคอมพิวเตอร์กับอุปกรณ์ภายนอก ลูกศรในรูปแสดงถึงทิศทางของข้อมูล การเชื่อมต่อแบบ Null modem หรือการเชื่อมต่อโดยตรงโดยไม่ต่องผ่านโมเด็ม โดยมีการตรวจสอบหรือแฮนด์เช็กเต็มรูปแบบ การเชื่อมต่อแบบ Null modem ในลักษณะที่ใช้สายสัญญาณเพียง 3 เส้น โดยเส้นหนึ่งสำหรับส่งข้อมูล อีกเส้นสำหรับรับข้อมูล และเส้นสุดท้ายเป็นกราวด์ สำหรับรายละเอียดหน้าที่การทำงานในแต่ละขาของพอร์ตอนุกรม RS-232 มีดังนี้

- *Data Carrier Detect : DCD หรืออาจเรียกว่า Carrier Detect : CD* ขานี้จะแอกตีฟเมื่อมีการส่งสัญญาณพาห์จากอุปกรณ์สื่อสารข้อมูลเช่น โมเด็ม สำหรับการใช้งานปกติ ขานี้จะไม่ได้ถูกใช้งานมากนัก

- Receive Data : RD หรือ RxD ขานี้ใช้เพื่อรับสัญญาณอนุกรมเข้ามายังคอมพิวเตอร์ โดยนำข้อมูลที่อ่านได้เก็บไว้ในรีจิสเตอร์ บัฟเฟอร์



รูปที่ 4-4 การจัดขาของคอนเน็กเตอร์อนุกรมตามมาตรฐาน RS-232 ทั้งแบบ DB-9 และ DB-25

รูปที่ 4-5 การต่ออุปกรณ์ภายนอกกับพอร์ตอนุกรมของคอมพิวเตอร์ในลักษณะต่าง ๆ

- Transmitted Data : TD หรือ TxD ใช้ส่งข้อมูลออกจากคอมพิวเตอร์ โดยนำข้อมูลที่เก็บอยู่ในบัฟเฟอร์สำหรับส่งข้อมูลส่งออกไป

- Data Terminal Ready : DTR เป็นขาสัญญาณที่ส่งออกจากคอมพิวเตอร์เพื่อให้อุปกรณ์ปลายทางรับรู้ว่าต้อง การติดต่อด้วยโดยขา DTR นี้ต้องเชื่อมต่อกับขา DSR ของอุปกรณ์ปลายทางและขา DTR ของอุปกรณ์ปลายทางต้องเชื่อมต่อกับขา DSR ของคอมพิวเตอร์ ถ้าใช้การเชื่อมต่อเป็นแบบ Null Modem ซึ่งใช้

สายในการเชื่อมต่อเพียง 3 เส้น จะต้องต่อขา DTR และ DSR ของตัวมันเองเข้าด้วยกันและต้องต่อกับขา DCD ด้วยในกรณีที่โปรแกรมสื่อสารที่ใช้มีการตรวจจับสัญญาณเพาห์

- Signal Ground · GND กราวด์ระบบ

- Data Set Ready · DSR ขานี้จะใช้คู่กับขา DTR เพื่อตรวจสอบการเชื่อมต่อกันระหว่างคอมพิวเตอร์กับอุปกรณ์ปลายทาง ซึ่งขา DSR นี้จะเป็นขาสำหรับรับข้อมูลจากภายนอกซึ่งถูกส่งมาจากขา DTR

- Request To Send RTS เป็นขาสำหรับส่งสัญญาณร้องขอให้ทางอุปกรณ์ปลายทางส่งข้อมูลกลับมายังคอมพิวเตอร์ โดยขาที่รับสัญญาณ RTS ก็คือขา CTS ในกรณีที่ใช้การเชื่อมต่อแบบ Null Modem 3 จะต้องเชื่อมต่อขา RTS และ CTS ของตัวมันเองด้วยกัน เพื่อจะให้การรับและส่งข้อมูลสามารถเกิดขึ้นได้ตลอดเวลา

- Clear To Send CTS ขานี้จะคอยรับสัญญาณจากขา RTS เมื่อรับสัญญาณได้ข้อมูลที่ขา TxD จะถูกส่งออกไป ดังนั้นขานี้จึงถูกใช้เพื่อตรวจสอบอุปกรณ์ต่อพ่วงว่าพร้อมที่จะรับข้อมูลหรือไม่

- Ring Indicator : RI ใช้แสดงสถานะสัญญาณเรียกจากสายโทรศัพท์ ปกติในการสื่อสารโดยทั่วไปสายนี้จะไม่ถูกใช้งาน จะใช้งานก็ต่อเมื่อมีการเชื่อมต่อกับโมเด็มและโปรแกรมมีการตรวจสอบสัญญาณนี้เท่านั้น

4.12.6 UART

UART มาจากคำว่า Universal Asynchronous Receiver Transmitter ซึ่งหมายถึงอุปกรณ์ที่ทำหน้าที่รับและส่งข้อมูลแบบอะซิงโครนัสนั่นเอง สำหรับการสื่อสารอนุกรมบนคอมพิวเตอร์แล้ว UART ถือว่าเป็นหัวใจสำคัญของการสื่อสารอนุกรม

หน้าที่หลักของUART คือ ทำหน้าที่แปลงข้อมูลที่อยู่ในรูปแบบขนานจากคอมพิวเตอร์ให้อยู่ในรูปแบบอนุกรมแบบอะซิงโครนัส แล้วส่งออกไปและทำหน้าที่แปลงสัญญาณอนุกรมแบบอะซิงโครนัสที่ป้อนเข้ามายัง UART ให้เป็นแบบขนานก่อนที่จะส่งเข้าสู่คอมพิวเตอร์ ซึ่งนอกจาก UART จะส่งข้อมูลไปยังคอมพิวเตอร์แล้ว ยังแจ้งข้อมูลอื่น ๆ ให้คอมพิวเตอร์รับทราบด้วย เช่น อัตราเร็วในการรับส่งข้อมูล (บอดเรต), รูปแบบการส่งข้อมูล, ความผิดพลาดที่เกิดขึ้นระหว่างการถ่ายทอดข้อมูล (ผิดพลาดจากพาริตี้, เฟรมข้อมูล, โอเวอร์รัน) เป็นต้น

ภายใน UART จะมีส่วนของวงจรสร้างบอดเรตแบบโปรแกรมได้ (programmable buadrate generator) โดยการกำหนดค่าตัวหารให้กับสัญญาณนาฬิกาของ UART โดยตัวหารนี้มีขนาด 16 บิต ดังนั้นจึงสามารถกำหนดตัวหารอยู่ในช่วง 1 – 65,535 UART สามารถรับส่งข้อมูลได้ทั้งแบบฮาล์ฟดูเพล็กซ์ (half duplex) และฟูลดูเพล็กซ์ (full duplex) โดยการส่งแบบฮาล์ฟดูเพล็กซ์เป็นการส่งแบบทิศทางเดียว ส่วนการส่งแบบฟูลดูเพล็กซ์นั้นสามารถรับและส่งข้อมูลได้ในคราวเดียวกัน

4.12.7 วงจรภายในและรีจิสเตอร์ของพอร์ตอนุกรม RS-232

เครื่องคอมพิวเตอร์โดยทั่วไปสามารถต่อพอร์ตอนุกรมสูงสุดได้ 4 พอร์ต มีชื่อเรียกเป็น com1, com2, com3 และ com4 ซึ่งพอร์ตอนุกรมแต่ละตัวต่างก็ใช้งาน UART ภายในคอมพิวเตอร์ในการติดต่อกับอุปกรณ์ภายนอกเช่นเดียวกัน การทำงานภายในของพอร์ตอนุกรมประกอบด้วยรีจิสเตอร์ 8 บิต 8 ตัวที่ใช้งานร่วมกับ UART แอดเดรสของรีจิสเตอร์ภายในพอร์ตอนุกรม สามารถคำนวณได้จากค่ารีจิสเตอร์พื้นฐานของพอร์ตอนุกรม *ยกตัวอย่าง* พอร์ตอนุกรม com1 มีแอดเดรสอยู่ที่ 3F8H ตำแหน่งของรีจิสเตอร์ต่าง ๆ จะเป็นตำแหน่งที่บวกเข้าไปกับค่า 3F8H โดยรีจิสเตอร์ที่ใช้งานกับพอร์ตอนุกรมมีดังนี้

00H เป็นรีจิสเตอร์บัฟเฟอร์สำหรับเก็บข้อมูลที่รับเข้ามาหรือเตรียมข้อมูลก่อนที่จะส่งออกไป

01H รีจิสเตอร์เอ็นเอเบิลการอินเตอร์รัปต์ ใช้เซตโหมดการอินเตอร์รัปต์ของพอร์ตอนุกรม

02H รีจิสเตอร์แสดงโหมดการอินเตอร์รัปต์ ใช้เพื่อตรวจสอบโหมดของการอินเตอร์รัปต์

03H รีจิสเตอร์กำหนดรูปแบบของข้อมูล

04H รีจิสเตอร์ควบคุมโมเด็ม ใช้ตรวจสอบบิตสำหรับติดต่อกับโมเด็ม เช่น RTS หรือ DTR

05H รีจิสเตอร์แสดงสถานการณ์รับและการส่งข้อมูลแบบอนุกรม

06H รีจิสเตอร์แสดงสถานะของโมเด็ม ซึ่งจะแสดงสถานะของขา DCD, RI, DSR และ CTS

07H รีจิสเตอร์สำหรับการเก็บข้อมูลชั่วคราว

4.12.8 ลักษณะสัญญาณอินพุตและเอาท์พุตของพอร์ต RS-232

สัญญาณเอาต์พุตที่ใช้ควบคุม (RTS และ DTR) และสัญญาณแสดงสถานะอินพุต (CTS, DSR และ DCD) ของพอร์ตอนุกรม RS-232 จะถูกกลับสถานะภายในตัว UART ส่วนสัญญาณข้อมูลทั้งภาคส่งและรับจะไม่ถูกกลับสถานะ UART จะให้ระดับสัญญาณเอาต์พุตออกมาเป็นแบบทีทีแอลเท่านั้น ดังนั้นเมื่อสัญญาณถูกส่งออกมาจาก UART จึงต้องส่งเข้าสู่วงจรขับเพื่อปรับระดับแรงดันให้ได้ระดับสัญญาณเป็นไปตามมาตรฐาน RS-232 ก่อนส่งออกไปจากคอมพิวเตอร์สำหรับอุปกรณ์ต่อเชื่อมปลายทางก็จะต้องมีวงจรขับในลักษณะนี้เช่นเดียวกัน เพื่อให้ได้ระดับสัญญาณในระดับเดียวกัน แต่วงจรขับที่ใช้ทั้งหมดภายในคอมพิวเตอร์และอุปกรณ์ต่อเชื่อมปลายทางนั้นจะถูกกลับสถานะ

แอดเดรสของพอร์ตอนุกรม

แอดเดรสพื้นฐานของพอร์ตอนุกรมมี 4 ตำแหน่งดังนี้คือ

COM1 : 3F8H

COM2 : 2F8H

COM3 3E8H

COM4 2E8H

บิต 3	บิต 2	บิต 1	จำนวนพอร์ต
0	0	0	ไม่มีพอร์ตอนุกรม
0	0	1	มีพอร์ตอนุกรม 1 พอร์ต
0	1	0	มีพอร์ตอนุกรม 2 พอร์ต
0	1	1	มีพอร์ตอนุกรม 3 พอร์ต
1	0	0	มีพอร์ตอนุกรม 4 พอร์ต

ตารางที่ 4-4 แสดงข้อมูลในแอดเดรส 0000 0411H ที่ใช้แจ้งจำนวนพอร์ตอนุกรม

เมื่อเริ่มเปิดเครื่องเพื่อใช้งานคอมพิวเตอร์ ไบออสภายในคอมพิวเตอร์จะทำการตรวจสอบแอดเดรสของพอร์ตอนุกรมทั้งหมด ถ้าไบออสตรวจพบแอดเดรสของพอร์ตอนุกรม ไบออสจะนำแอดเดรสที่ตรวจพบไปเก็บไว้ในหน่วยความจำขนาด 2 ไบต์ สำหรับพอร์ตอนุกรม COM1 จะเก็บไว้ที่แอดเดรส 0000:0400H และ 0000:0401H ส่วนตำแหน่งอื่น ๆ มีรายละเอียดดังนี้

COM2 = 0000:0402H – 0000:0403H

COM3 = 0000:0404H – 0000:0405H

COM4 = 0000:0406H – 0000:0407H

นอกจากนี้ที่หน่วยความจำแอดเดรส 0000:0411H ยังใช้สำหรับแสดงจำนวนของพอร์ตอนุกรมที่มีอยู่ในคอมพิวเตอร์อีกด้วย

4.12.9 การเขียนโปรแกรมเพื่อการใช้งานพอร์ตอนุกรม

ก่อนการใช้งานพอร์ตอนุกรมนั้นจะต้องมีการกำหนดค่าเริ่มต้นให้กับตัวมันก่อน ซึ่งก็คือการกำหนดจำนวนบิตข้อมูลที่ต้องการส่ง,จำนวนบิตปิดท้าย,ชนิดของพาริตี้ที่ใช้ และบอดเรต การกำหนดสามารถทำได้หลายวิธี วิธีแรกเป็นการกำหนดจากดอสพร้อมพ์ โดยใช้คำสั่ง MODE ซึ่งมีวิธีการใช้งานดังนี้

แต่เมื่อใช้คำสั่งนี้ในขณะที่โปรแกรมทำงานผ่านระบบปฏิบัติการวินโดวส์ จะไม่สามารถใช้งานได้ เนื่องจากระบบปฏิบัติการวินโดวส์ ได้เข้าผังตัวพอร์ตอนุกรมเข้าเป็นส่วนหนึ่งของระบบปฏิบัติการแล้ว ดังนั้นการเรียกใช้งานจึงจำเป็นต้องเรียกผ่านเครื่องมือที่ติดต่อผ่านระบบปฏิบัติการ เช่นการใช้คอนโทรล MSCOMM32.OCX ของโปรแกรม Visual BASIC

4.12.10 คอนโทรล MSComm

สำหรับการใช้งาน Visual BASIC ตั้งแต่เวอร์ชัน 2 เป็นต้นมา ใน Visual BASIC จะมีคัสตอมคอนโทรลสำหรับการสื่อสารอนุกรมผ่านทางพอร์ตอนุกรมของคอมพิวเตอร์มาให้ โดยใน Visual BASIC เวอร์ชัน 2 และเวอร์ชัน 3 จะใช้ชื่อว่า MSCOMM.VBX ส่วนเวอร์ชัน 4 ใช้ชื่อว่า MSCOMM16.OCX สำหรับการทำงานกับระบบปฏิบัติการ 16 บิต และ MSCOMM32.OCX สำหรับการทำงานกับระบบปฏิบัติการ 32 บิต สำหรับใน Visual BASIC เวอร์ชัน 5 จะมีเพียง MSCOMM 32.OCX เท่านั้นเพราะถูกออกแบบมาให้ใช้งานกับระบบปฏิบัติการ 32 บิต

MSComm จัดเตรียมทางเลือกเอาไว้ 2 ทางเพื่อความสะดวกในการสื่อสารข้อมูล ทางแรกคือ การสื่อสารข้อมูลที่กระตุ้นด้วยเหตุการณ์ (event-driven communications) เป็นรูปแบบการใช้งานที่มีประสิทธิภาพมากสำหรับการตอบสนองแบบทันทีทันใด เช่น เมื่อตัวอักษรถูกส่งมาที่พอร์ตอนุกรมหรือเกิดการเปลี่ยนแปลงที่ขา Data Carrier Detect (DCD) หรือขา Request To Send (RTS) เหตุการณ์ Oncomm ของ MSComm จะสามารถตรวจจับสัญญาณนั้นได้ทันที ซึ่งจะกล่าวถึงรายละเอียดในหัวข้อคุณสมบัติ *CommEvent* ต่อไป ส่วนทางเลือกที่สองเป็นการคอยตรวจสอบค่าเหตุการณ์และความผิดพลาดที่เกิดขึ้นด้วยการดูค่าที่เปลี่ยนแปลงภายในคุณสมบัติ CommEvent หลังจากให้โปรแกรมทำงานในฟังก์ชั่นต่าง ๆ ไปเรียบร้อยแล้ว ซึ่งวิธีนี้ใช้งานได้ดีในกรณีที่โปรแกรมมีขนาดเล็ก

คอนโทรล MSComm 1 ตัวสามารถควบคุมการทำงานพอร์ตอนุกรมได้ 1 พอร์ต ถ้าในโปรแกรมที่ใช้งานต้องการติดต่อกับพอร์ตอนุกรมมากกว่า 1 พอร์ตจะต้องใช้คอนโทรล MSComm มากกว่า 1 ตัวเพื่อควบคุมพอร์ตอนุกรมในแต่ละพอร์ต แอดเดรสของพอร์ตอนุกรมและแอดเดรสของการเกิดอินเตอร์รัปต์สามารถเปลี่ยนแปลงได้จากการแก้ไขค่าที่ Control Panel

ถึงแม้ว่า คอนโทรล MSComm จะมีคุณสมบัติ (property) มากมายแต่สามารถทำความเข้าใจได้ไม่ยาก

CommPort

ใช้ในการกำหนดและอ่านค่าพอร์ตอนุกรมที่ติดต่ออยู่ (COM1, COM2, COM3, COM4)

รูปแบบการใช้งาน

Object.Settings [=value]

ค่า Value มีชนิดข้อมูลเป็นแบบ String มีรูปแบบเป็น "BBBB, P, D, S" โดย BBBB เป็นค่าอัตราบอด, P เป็นค่าพาริตี้, D เป็นจำนวนของบิตข้อมูล และ S เป็นจำนวนของบิตปิดท้าย ปกติแล้วค่านี้ถูกกำหนดไว้เป็น "9600, N, 8, 1"

ตัวอย่างการใช้งานคำสั่ง Settings โดยจะเป็นการกำหนดค่าบอดเรตเท่ากับ 9600 ไม่มีพาริตี้ จำนวนบิตข้อมูล 8 บิต และ บิตปิดท้าย 1 บิต สามารถเขียนโปรแกรมได้ดังนี้

MSComm1.Settings = "9600, N, 8, 1"

PortOpen

ใช้ในการกำหนดและอ่านค่าสถานะของพอร์ตอนุกรม เพื่อเปิดและปิดพอร์ตอนุกรม

รูปแบบการใช้งาน

Object.PortOpen [= value]

ค่า Value มีชนิดข้อมูลเป็นแบบบูลีน คือ True กับ False โดย True หมายถึงการเปิดพอร์ตอนุกรมและ False หมายถึงการปิดพอร์ตอนุกรม สำหรับการปิดพอร์ตนั้นจะมีการเคลียร์บัฟเฟอร์รับข้อมูลและบัฟเฟอร์ส่งข้อมูลด้วย คอนโทรล MSComm จะปิดพอร์ตอนุกรมโดยอัตโนมัติเมื่อออกจากโปรแกรม ก่อนที่จะใช้คุณสมบัติ PortOpen ต้องตรวจสอบให้แน่ใจก่อนว่าคุณสมบัติ CommPort นั้นได้ทำการกำหนดตำแหน่งของพอร์ตอนุกรมไว้ถูกต้องหรือไม่ มิเช่นนั้น MSComm จะแสดงข้อผิดพลาด Error 68 แจ้งแก่ผู้ใช้งาน หรือถ้าพอร์ตอนุกรมนั้นถูกเปิดเอาไว้แล้ว โปรแกรมก็จะแจ้งข้อผิดพลาดออกมาเช่นเดียวกัน

ถ้าคุณสมบัติ *DTREnable* หรือ *RTSEnable* ถูกกำหนดให้เป็น *True* ก่อนที่จะทำการเปิดพอร์ต ค่าคุณสมบัติของ *DTREnable* หรือ RTSEnable จะถูกเซตเป็น False หลังจากปิดพอร์ต แต่ถ้าเซตเป็น False หลังจากปิดโปรแกรมแล้ว ค่าที่กำหนดไว้จะเป็นค่าเดิม

ตัวอย่างการใช้คำสั่งเปิดพอร์ต เพื่อติดต่อสื่อสารกับพอร์ตอนุกรม COM1 และมีบอดเรต 9600 บิตต่อวินาที ไม่มีพาริตี้ จำนวนบิตข้อมูล 8 บิต และบิตปิดท้าย 1 บิต มีดังนี้

```
MSComm.Settings = "9600, n, 8, 1"
MSComm.CommPort = 1
MSComm.PortOpen = True
```

Input

อ่านค่าและลบค่าขบวนข้อมูลจากบัฟเฟอร์ภาครับ

รูปแบบการใช้งาน

Object.Input

คุณสมบัติ InputLen เป็นตัวกำหนดจำนวนของตัวอักษรที่จะอ่านโดยคุณสมบัติ Input การกำหนดค่าให้ InputLen เท่ากับ 0 เป็นการกำหนดให้คุณสมบัติ Input ทำการอ่านค่าข้อมูลในบัฟเฟอร์รับข้อมูลทั้งหมด

คุณสมบัติ InputMode เป็นตัวกำหนดชนิดของข้อมูลที่คุณสมบัติ Input รับเข้ามา ถ้าInputMode ถูกกำหนดเป็น comInputModeText คุณสมบัติ Input จะส่งค่าข้อมูลกลับมาในรูปของข้อความชนิดข้อมูลเป็นแบบ Variant ถ้า InputMode กำหนดเป็น comInputModeBinary คุณสมบัติ Input จะส่งข้อมูลกลับมาในรูปของไบนารี่และชนิดข้อมูลเป็นแบบ Variant

ตัวอย่างโปรแกรมแสดงให้เห็นถึงวิธีในการรับข้อมูลจากบัฟเฟอร์รับข้อมูลทั้งหมด

```
Private Sub Command1_Click ( )
Dim InString as String
        MSComm1.InputLen = 0          ' Retrieve all available data.
        If MSComm1.InBufferCount Then   ' Check for data.
        InString = MSComm.Input              ' Read data.
        End If
End Sub
```

InBufferCount

ส่งค่าจำนวนของตัวอักษรที่อยู่ในบัฟเฟอร์กาครับ

รูปแบบการใช้งานคำสั่ง

Object.InBufferCount[= value]

คำสั่ง InBufferCount จะแสดงค่าจำนวนของตัวอักษร ซึ่งรับมาจากภายนอกและยังเก็บอยู่ในบัฟเฟอร์ภาครับเพื่อให้ผู้ใช้งานอ่านค่าออกไป สำหรับการเคลียร์ค่าบัฟเฟอร์ภาครับทำได้โดยกำหนดให้ InBufferCount มีค่าเป็น 0

InBufferSize

กำหนดและคืนค่าขนาดของบัฟเฟอร์ภาครับในหน่วยเป็นไบต์

รูปแบบการใช้งานคำสั่ง

Object.InBufferSize [= value]

คำสั่ง InBufferSize ใช้เพื่อกำหนดขนาดของบัฟเฟอร์ภาครับ ค่าเริ่มต้นกำหนดไว้ที่ 1024 ไบต์

InputLen

กำหนดค่าและคืนค่าจำนวนของตัวอักษรที่อ่านจากบัฟเฟอร์ภาครับ

รูปแบบการใช้งานคำสั่ง

Object.InputLen [= value]

ค่าเริ่มต้นของคุณสมบัติ InputLen มีค่าเท่ากับ "0" การกำหนดค่าเท่ากับ "0" จะทำให้ คำสั่ง Input ของ MSComm อ่านค่าข้อมูลที่อยู่ภายในบัฟเฟอร์ภาครับทั้งหมด

ถ้าไม่มีข้อมูลอยู่ในบัฟเฟอร์ภาครับมากเท่ากับจำนวน InputLen คำสั่ง Input จะส่งค่าว่าง (" ") กลับออกมา ผู้ใช้งานสามารถตรวจสอบข้อมูลในบัฟเฟอร์ภาครับได้โดยใช้คุณสมบัติ InBufferCount โดยกำหนดให้มีข้อมูลอยู่ในบัฟเฟอร์ภาครับก่อนแล้วจึงค่อยอ่านข้อมูลจากบัฟเฟอร์ภาครับ

คุณสมบัตินี้มักใช้กับการอ่านค่าข้อมูลจากเครื่องมือหรือเครื่องจักรที่มีการกำหนดค่าขนาดความยาวของข้อมูลเอาไว้แล้ว

InputMode

กำหนดค่าและคืนค่าชนิดของข้อมูลที่รับโดยคำสั่ง Input

รูปแบบการใช้งานคำสั่ง

Object.InputMode [= value]

คุณสมบัติ InputMode ใช้กำหนดว่าข้อมูลชนิดไหนที่รับเข้ามาผ่านคำสั่ง Input โดยมี 2 ประเภทคือ

ComInputModeText สำหรับข้อมูลที่อยู่ในรูปข้อความตัวอักษรตามมาตรฐาน ANSI โดยจะต้องกำหนดค่าเป็น "0" และค่าเริ่มต้นของการรับค่าข้อมูลก็จะเป็นค่านี้

ComInputModeBinary สำหรับข้อมูลอื่น ๆ ซึ่งจะเก็บในรูปไบนารี่รวมกันอยู่เป็นไบต์ข้อมูลและเก็บข้อมูลไว้ในตัวแปรแบบอาเรย์ ชนิดข้อมูลเป็นแบบไบต์

Output

ใช้ในการส่งขบวนการของข้อมูลไปยังบัฟเฟอร์ส่งข้อมูล

รูปแบบการใช้งาน

Object.Output [= value]

ค่า value เป็นค่าของตัวอักษรที่เขียนไปยังบัฟเฟอร์ส่งข้อมูล คุณสมบัติ Output สามารถใช้ในการส่งข้อมูลตัวอักษรหรือข้อมูลไบนารี่ก็ได้ โดยการส่งข้อมูลเป็นรูปแบบตัวอักษรจะต้องกำหนดข้อมูลเป็นแบบ Variant และมีข้อมูลภายในเป็นแบบ String สำหรับการส่งข้อมูลไบนารี่จะต้องกำหนดชนิดของข้อมูลเป็นแบบ Variant และมีข้อมูลภายในเป็นแบบ Byte

OutBufferCount

คืนค่าจำนวนของข้อมูลตัวอักษรที่เก็บอยู่ในบัฟเฟอร์ภาคส่งและสามารถใช้คำสั่งนี้เพื่อเคลียร์บัฟเฟอร์ภาคส่งได้ด้วย

รูปแบบการใช้งานคำสั่ง

Object.OutBufferCount [= value]

ผู้ใช้งานสามารถเคลียร์บัฟเฟอร์ภาคส่งได้โดยการกำหนดค่า OutBufferCount เท่ากับ "0"

OutBufferSize

กำหนดค่าและคืนค่าขนาดของบัฟเฟอร์ภาคส่ง ชนิดตัวแปรเป็นแบบไบต์

รูปแบบการใช้งานคำสั่ง

Object.OutBufferSize [= object]

คุณสมบัติ OutBufferSize ใช้สำหรับกำหนดขนาดของบัฟเฟอร์ภาคส่ง โดยค่าปกติที่ใช้งานจะมีค่าเท่ากับ 512 ไบต์

ParityReplace

กำหนดและคืนค่าตัวอักษรที่ไปวางแทนในตำแหน่งที่เกิดข้อผิดพลาดจากพาริตี้

รูปแบบการใช้งานคำสั่ง

Object.ParityReplace [= value]

บิตพาริตี้ เป็นบิตที่ทางภาคส่งข้อมูลทำการส่งมาพร้อมกับข้อมูล เพื่อตรวจสอบข้อผิดพลาดของข้อมูล โดยเมื่อมีการใช้บิตพาริตี้ คอนโทรล MSCOM จะทำการบวกบิตทุกบิตที่มีค่าลอจิก "1" ในแต่ละไบต์ และทำการตรวจสอบผลลัพธ์ว่าบิตที่อ่านได้นั้นมีจำนวนลอจิก "1" เป็นเลขคู่หรือคี่ และตรงกับค่าที่กำหนดไว้แต่ต้นหรือไม่ ถ้าค่าที่นำมาบวกแล้วมีพาริตี้ไม่ตรงแสดงว่าการรับส่งข้อมูลผิดพลาด

การกำหนดค่า เริ่มต้นให้กับ ParityReplace นั้นกำหนดให้ใช้เครื่องหมาย (?) ไปวางไว้ที่ตำแหน่งที่เกิดพาริตี้ผิดพลาด ถ้ากำหนดค่า ParityReplace ให้เป็นค่าว่าง (" ") จะเป็นการยกเลิกการใช้งาน ParityReplace ใช้ชนิดข้อมูลเป็นแบบสตริง แต่จะทำการกำหนด จะกำหนดได้เพียงไบต์เดียวเท่านั้น ซึ่งจะสามารถใช้ค่าใด ๆ ก็ได้ที่เป็นโค้ด ANSI มีค่าอยู่ระหว่าง 0-255

DTREnable

ใช้ในการกำหนดสถานะลอจิกของขา Data Terminal Ready (DTR) โดยสัญญาณของขา DTR จะส่งจากคอมพิวเตอร์ไปยังโมเด็มเพื่อแสดงว่าคอมพิวเตอร์พร้อมที่จะรับข้อมูลแล้ว ชนิดของข้อมูลเป็นแบบบูลีน

รูปแบบการใช้งาน

Object.DTREnable [= value]

ค่า Value เป็นสถานะ True หรือ False เพื่อกำหนดลอจิกของขา DTR ให้เป็น "0" หรือ "1" โดย

True หมายถึง ให้ขา DTR มีลอจิก "1"

False หมายถึง ให้ขา DTR มีลอจิก "1"

สำหรับการใช้งานกับโมเด็ม การทำให้ขา DTR เป็นลอจิก "0" จะเป็นการวางหูโทรศัพท์หรือยกเลิกการติดต่อ

RTSEnable

ใช้เพื่อกำหนดสถานะลอจิกให้ขา Request To Send (RTS) โดยขา RTS จะเป็นสัญญาณที่ส่งจากคอมพิวเตอร์ไปยังโมเด็มเพื่อร้องขอส่งข้อมูล ชนิดของข้อมูลเป็นแบบ Boolean

รูปแบบการใช้งาน

Object.RTSEnable [= value]

ค่า Value เป็นสถานะ True หรือ False เพื่อกำหนดลอจิก "0" หรือ "1" ให้ขา RTS โดย

True หมายถึง ให้ขา RTS มีลอจิก "1"

False หมายถึง ให้ขา RTS มีลอจิก "0" (เป็นค่าปกติ)

EOFEnable

เป็นการกำหนดให้ MSComm รอสัญลักษณ์แสดงส่วนท้ายสุดของไฟล์ (End of file : EOF) ระหว่างการรับอินพุตเข้ามา ถ้าพบสัญลักษณ์ EOF ภาคอินพุตจะหยุดรับข้อมูล และเหตุการณ์ OnComm จะถูกกระตุ้นให้ทำงาน คุณสมบัติ commEvent จะมีค่าเท่ากับ 7 หรือ ComEvEOF

รูปแบบการใช้งาน

Object.EOFEnable [= value]

โดย value เป็นค่าสถานะ True หรือ False เพื่อเอ็นเอเบิลหรือดิสเอเบิลการทำงานของเหตุการณ์ OnComm เมื่อตรวจพบสัญลักษณ์ EOF โดย

True หมายถึง เหตุการณ์ OnComm จะถูกกระตุ้นให้ทำงานด้วย EOF

False หมายถึง เหตุการณ์ OnComm จะถูกกระตุ้นให้ทำงานด้วย EOF (เป็นค่าปกติ)เมื่อ EOFEnable กำหนดให้เป็น False ส่วนควบคุมจะไม่มีการตรวจสอบสัญลักษณ์ EOF

CTSHolding

ผู้ใช้งานสามารถตรวจสอบการทำงานของขา Clear To Send (CTS) ได้ว่ามีสถานะลอจิก "0" หรือ "1" โดยค่าที่อ่านได้จะเป็นบูลีน True และ False ถ้าค่า CTSHolding เป็น True ขา CTS จะมีสถานะลอจิกเป็น "1" ถ้าค่า CTSHolding เป็น False ขา CTS จะมีสถานะลอจิกเป็น "0"

รูปแบบการใช้งาน

Object.CTSHolding

เมื่อขา CTS เป็นลอจิก "0" (CTSHolding = False) และเกิดไทม์เอาต์ คอนโทรล MSComm จะกำหนดให้คุณสมบัติ CommEvent มีค่าเป็น comEventCTSTO (Clear To Send Timeout) และกระตุ้นให้เกิดเหตุการณ์ OnComm

CDHolding

ผู้ใช้งานสามารถตรวจสอบการทำงานของขา Data Carrier Detect (DCD) ได้ว่ามีสถานะลอจิกเป็น "1" หรือ "0" โดยค่าที่อ่านได้จะเป็นบูลีน True และ False ถ้าค่า CDHolding เป็น True ขา DCD จะมีสถานะลอจิก "1" หรือ "0" โดยค่าที่อ่านได้จะเป็นบูลีน True และ False ถ้าค่า DSRHolding เป็น True ขา DSR จะมีสถานะลอจิก "1" ถ้าค่า DSRHolding เป็น False ขา DSR จะมีสถานะลอจิก "0"

รูปแบบการใช้งาน

Object.DSRHolding

เมื่อขา DSR เป็นลอจิก "1" (DSRHolding = True) และเกิดไทม์ คอนโทรล MSComm จะกำหนดให้คุณสมบัติ CommEvent มีค่าเป็น comEventDSRTO (Data Set Ready Timeout) และกระตุ้นให้เกิดเหตุการณ์ *OnComm*

รูปแบบการใช้งานคำสั่ง

Object.Handshaking [= value]

ค่าตัวแปร Value ที่ใช้กำหนดค่ากำหนดได้ 4 รูปแบบด้วยกันคือ

1. comNone ค่าที่กำหนดคือ 0 เป็นการกำหนดให้ไม่มีการแฮนด์เช็ก (เป็นค่าเริ่มต้น)

2. comXOnXOff ค่าที่กำหนดคือ 1 เป็นการกำหนดให้ใช้แฮนด์เช็กแบบ XON/XOFF

3 commRTS ค่าที่กำหนดคือ 2 เป็นการกำหนดให้ใช้ขา RTS/CTS (Request To Send/Clear To Send)

4. comRTSXOnXOff ค่าที่กำหนดคือ 3 เป็นการกำหนดให้ใช้ทั้งแบบ Request To Send และ XON/XOFF

คุณสมบัติ Handshaking ใช้เพื่อกำหนดรูปแบบการสื่อสารภายใน ระหว่างที่ข้อมูลถูกส่งไปยังบัฟเฟอร์ภาครับ เมื่อข้อมูลตัวอักษรถูกส่งมาถึงพอร์ตอนุกรม อุปการณ์สื่อสารข้อมูลจะทำการย้ายข้อมูลไปยังบัฟเฟอร์ภาครับ เพื่อที่จะให้โปรแกรมสามารถอ่านค่าไปใช้งานได้ ถ้าไม่มีบัฟเฟอร์ภาครับ โปรแกรมที่ใช้งานจะต้องทำการอ่านค่าข้อมูลโดยตรงจากฮาร์ดแวร์ของพอร์ตอนุกรม ซึ่งผู้ใช้งานจะเกิดปัญหาข้อมูลสูญหายได้ เนื่องจากว่าการเปลี่ยนแปลงของข้อมูลที่ส่งเข้ามามีการเปลี่ยนแปลงอย่างรวดเร็ว

คุณสมบัติ handshaking ช่วยให้ผู้ใช้งานแน่ใจได้ว่าข้อมูลที่ได้รับมานั้นไม่มีการสูญหายเมื่อบัฟเฟอร์ภาครับที่รับข้อมูลนั้นเกิดข้อมูลล้นหรือโอเวอร์โฟล (overflow) โดยใช้วิธีการตรวจสอบความพร้อมของบัฟเฟอร์ว่าพร้อมรับข้อมูลหรือไม่ก่อนที่จะส่งข้อมูลมาให้

รูปแบบการใช้งาน

Object.Break [= value]

เหตุการณ์ OnComm

เหตุการณ์ OnComm จะถูกสร้างขึ้นเมื่อค่าของคุณสมบัติ CommEvent มีการเปลี่ยนแปลงเพื่อแสดงผลการเปลี่ยนแปลงเหล่านั้นแบบทันทีทันใดหรือ แสดงข้อผิดพลาดที่เกิดขึ้น ตัวอย่างโปรแกรมย่อย OnComm เพื่อนำเหตุการณ์ CommEvent มาแสดงมีดังนี้

การใช้ MSComm เพื่อการติดต่อฮาร์ดแวร์

จากรายละเอียดของ MSComm ที่กล่าวไปในตอนต้นนั้น จะเห็นได้ว่าวิธีการที่จะอ่านค่าหรือเขียนค่าไปยังขาสถานะและขาควบคุมของพอร์ตอนุกรมสามารถทำได้ง่ายดายมาก โดยใช้คำสั่งเหล่านี้

DTREnable	สำหรับสั่งให้ขา DTR มีลอจิก "0" หรือ "1"
RTSEnble	สำหรับสั่งให้ขา RTS มีลอจิก "0" หรือ "1"
CTSHolding	สำหรับอ่านค่าสถานะจากขา CTS ว่ามีลอจิก "0" หรือ "1"
CDHolding	สำหรับอ่านค่าสถานะจากขา DCD ว่ามีลอจิก "0" หรือ "1"
DSRHolding	สำหรับอ่านค่าสถานะจากขา DSR ว่ามีลอจิก "0" หรือ "1"
Break	สำหรับการสั่งให้ขา TxD มีลอจิก "0" หรือ "1"

4.13 พอร์ต USB กับวิชวลเบสิก

USB เป็นการส่งข้อมูลที่มีรูปแบบการเชื่อมต่อในระบบบัสคือ อุปกรณ์ทุกๆ ตัวจะต้องส่งสัญญาณรวมกันไปในสายส่งสัญญาณเพียงคู่เดียว ดังนั้นดอุปกรณ์ทุกๆ ตัวที่เชื่อมต่อกับบัสจะต้องส่งข้อมูลเรียงลำดับกันไปเพื่อไม่ให้เกิดการชนกันของข้อมูล และเนื่องจาก USB เป็นระบบบัสที่ใช้สายส่งสัญญาณเพียงคู่เดียว (2 เส้น) ทำให้ในช่วงเวลาหนึ่งๆ จะมีข้อมูลวิ่งไปได้เพียงทิศทางเดียวเท่านั้น ไม่สามารถเกิดการรับและส่งข้อมูลไปในเวลาเดียวกันได้หรือที่เรียกกันว่า การส่งข้อมูลแบบฮาล์ฟดูเพล็กซ์ (half duplex)

เนื่องจากการถ่ายยอดข้อมูลในบัส USB เกิดขึ้นด้วยความเร็วสูง ทำให้ต้องนำเทคนิคการถ่ายทอดสัญญาณรูปแบบต่างๆ มาใช้งานเพื่อให้ข้อมูลส่งไปยังปลายทางได้ถูกต้อง การถ่ายทอดสัญญาณแบบผลต่างเป็นเทคนิคที่นำมาใช้งานเพื่อให้ข้อมูลส่งไปยังปลายทางได้ถูกต้อง การถ่ายทอดสัญญาณแบบผลต่างเป็นเทคนิคที่นำมาใช้เพื่อลดคลื่นรบกวนที่จะแพร่ออกมาจากสายดังที่กล่าวไปแล้ว นอกจากนั้นยังต้องอาศัยการเข้ารหัสแบบ NRZI เพื่อให้ด้านรับทราบถึงจังหวะข้อมูลในแต่ละบิตเพื่อไม่ให้เกิดความผิดพลาด และสามารถตรวจสอบการเชื่อมต่อหรือปลดออกของอุปกรณ์ในบัส จึงต้องมีการกำหนดรูปแบบของสัญญาณต่างๆ เพื่อใช้งานขึ้น

4.13.1 การส่งข้อมูลในสายสัญญาณ

หลังจากข้อมูลถูกเข้ารหัสเรียบร้อยแล้ว ข้อมูลเหล่านั้นก็จะถูกส่งออกไปยังอุปกรณ์ต่างๆที่ต่ออยู่กับฮับ แต่เนื่องจาก USB ส่งข้อมูลด้วยความเร็วที่สูงมาก โดยใช้สัญญาณที่ยาว (เมื่อเทียบกับสายสัญญาณข้อมูลภายในคอมพิวเตอร์) ทำให้ไม่สามารถใช้การส่งสัญญาณแบบขั้วเดียว (single end) ได้เพราะจะทำให้เกิดการแพร่กระจายของคลื่นแม่เหล็กไฟฟ้าไปกวนอุปกรณ์อื่นๆ และยังทำให้สัญญาณรบกวนจากภายนอกเข้าไปกวนข้อมูลที่ต้องการส่งในสายสัญญาณได้ง่าย

การส่งสัญญาณที่แก้ไขปัญหา 2 ข้อที่กล่าวมาก็คือการส่งสัญญาณผลต่าง (differential pair signaling) เนื่องจากการส่งสัญญาณด้วยวิธีนี้จะทำให้สนามแม่เหล็กไฟฟ้าที่เกิดขึ้นจากสายสัญญาณทั้งสองเส้นหักล้างกัน เนื่องจากมีศักย์ตรงข้ามกัน จึงไม่เกิดการแพร่กระจายไปรบกวนอุปกรณ์อื่นๆ สัญญาณ 2 เส้นอยู่คู่กัน เมื่อสัญญาณรบกวนเหนี่ยวนำเข้าสู่สายเส้นหนึ่งก็จะเหนี่ยวนำเข้าสู่สายอีกเส้นในขนาดที่เท่ากัน เมื่อถึงปลายทางในวงจรภาครับจะใช้วงจรขยายผลต่าง (differential amplifier) เป็นตัวรับสัญญาณ ซึ่งวงจรนี้จะมีคุณสมบัติขยายสัญญาณที่มีขนาดต่างกันและลดทอนสัญญาณที่มีขนาดเท่ากัน ดังนั้นสัญญาณข้อมูลซึ่งมีศักย์ตรงข้ามกันจะถูกขยายเพื่อส่งต่อไปยังส่วนถัดไป แต่สัญญาณรบกวนซึ่งมีขนาดเท่ากันในสายทั้ง 2 เส้นจะถูกลดทอนหรือตัดทิ้ง

จะเห็นได้ว่าทั้งสองด้านมีวงจรที่เหมือนกันเนื่องจาก USB เป็นบัสที่รับส่งข้อมูลแบบฮาล์ฟดูเพล็กซ์ นั่นคือทั้งสองด้านสามารถรับส่งข้อมูลได้ทั้งคู่ แต่คนละช่วงเวลากัน ทำให้ทั้งสองด้านมีชุดวงจรที่เหมือนกัน และเนื่องจากการรับส่งข้อมูลแบบนี้จะมีอุปกรณ์เพียงตัวเดียวที่สามารถรับหรือส่งข้อมูลได้ในช่วงเวลาหนึ่งๆ อุปกรณ์ที่เหลือที่ไม่ได้ใช้งาน บัสจำเป็นจะต้องปล่อยบัสสัญญาณให้มีสภาพอิมพีแดนซ์สูง (high impedance) ดังนั้นวงจรภาครับส่งของอุปกรณ์แต่ละตัวจะต้องสามารถกำหนดสถานะของเอาต์พุตให้เป็นสภาพอิมพีแดนซ์สูงได้

4.13.2 HID มาตรฐานของอุปกรณ์ USB ในระดับเชื่อมต่อกับผู้ใช้งาน

HID Class (Human Interface Device Class) เป็นลักษณะการเชื่อมต่อของอุปกรณ์ USB ที่เน้นไปที่การติดต่อกับมนุษย์ (human) หรือผู้ใช้งานนั่นเอง ซึ่งอุปกรณ์ USB ที่ใช้ระดับการเชื่อมต่อแบบนี้ซึ่งรู้จักกันดี คือ คีย์บอร์ดและเมาส์ ซึ่งจะอธิบายถึงรูปแบบ มาตรฐาน ลักษณะการเชื่อมต่อและถ่ายทอดข้อมูล เนื่องจากในส่วนของการทดลองจะใช้วงจรเชื่อมต่อพอร์ต USB ที่ใช้มาตรฐานการเชื่อมต่อในระดับ HID นี้ ดังนั้นการทำความรู้จักกับ HID จึงเป็นสิ่งที่ผู้สนใจเชื่อมต่อคอมพิวเตอร์กับอุปกรณ์ภายนอกผ่านพอร์ต USB ควรทราบกรอบกับระดับการเชื่อมต่อแบบนี้เป็นระดับการเชื่อมต่อที่สามารถทำความเข้าใจได้ง่ายและเป็นที่นิยมนำมาใช้สร้างอุปกรณ์ USB มากที่สุด

4.13.2.1 สิ่งที่ควรรู้เกี่ยวกับ HID คลาส

1. การเปลี่ยนแปลงของข้อมูลที่เกิดขึ้นในโครงสร้างการทำงานจะเรียกว่า รายงานหรือรีพอร์ตตามข้อตกลงกำหนดของ HID รีพอร์ต โดยโฮสต์จะทำการรับส่งข้อมูลด้วยการส่งและร้องขอรีพอร์ตในการควบคุมหรือมีลักษณะเป็นการถ่ายทอดสัญญาณแบบอินเตอร์รัปต์ อย่างไรก็ตามรูปแบบของรีพอร์ตนี้ยังไม่มีข้อมูลสรุปชัดเจน แต่ก็มีความอ่อนตัวสูง ทำให้สามารถรองรับกับข้อมูลได้ทุกรูปแบบ

2. อุปกรณ์ USB สามารถส่งข้อมูลกลับไปยังโฮสต์ (ซึ่งส่วนใหญ่คือ คอมพิวเตอร์) ได้ตลอดเวลา โดยไม่สามารถคาดหมายล่วงหน้า ดังนั้นจึงเป็นหน้าที่ของโฮสต์เองที่ต้องคอยวนตรวจสอบอยู่ตลอดเวลา เพื่อให้สามารถรองรับกับการทำงานของอุปกรณ์ได้ทันท่วงที

3. ในการถ่ายทอดสัญญาณหรือข้อมูลบนบัสของ USB ไม่สามารถทราบถึงอัตราการส่งผ่านข้อมูลได้อย่างแน่นอน สิ่งที่กำหนดได้มีเพียงคาบเวลาในแต่ละทรานแซกชั่น และค่าเวลาระหว่างทราบแซกชั่นนั้นอาจน้อยกว่าก็ได้ อย่างไรก็ตาม ระบบจะพยายามทำให้เกิดอัตราการส่งผ่านข้อมูลเร็วที่สุดเท่าที่จะเป็นไปได้

4. อัตราการถ่ายทอดสัญญาณหรือข้อมูลสูงสุดจะถูกจำกัดไว้ตามประเภทของอุปกรณ์ ซึ่งมีด้วยกัน 3 แบบคือความเร็วต่ำ, ความเร็วเต็มที่ และความเร็วสูง โดยโฮสต์สามารถกำหนดคาบเวลาในการทำงานต่อทรานแซกชั่นสูงสุดแยกกันไปตามประเภทของอุปกรณ์ โดย

4.1 สำหรับอุปกรณ์ความเร็วต่ำ โฮสต์กำหนดคาบเวลาในการทำงาน 1 ทรานแซกชั่นสูงสุดไม่เกิน 10 มิลลิวินาที ทำให้อัตราการถ่ายทอดสัญญาณเกิดขึ้นได้สูงสุดไม่เกิน 800 ไบต์ต่อวินาที

4.2 สำหรับอุปกรณ์ความเร็วต่ำ โฮสต์กำหนดคาบเวลาในการทำงาน 1 ทรานแซกชั่นสูงสุดไม่เกิน 1 มิลลิวินาที ทำให้อัตราการถ่ายทอดสัญญาณเกิดขึ้นได้สูงสุดไม่เกิน 8,000 ไบต์ต่อวินาที

4.2 สำหรับอุปกรณ์ความเร็วต่ำ โฮสต์กำหนดคาบเวลาในการทำงาน 3 ทรานแซกชั่นสูงสุดไม่เกิน 125 ไมโครวินาที ทำให้อัตราการถ่ายทอดสัญญาณเกิดขึ้นได้สูงสุดไม่เกิน 24.56 เมกะไบต์ต่อวินาที

5. ในระบบปฏิบัติการวินโดวส์ที่ต่ำกว่า 98SE จะไม่รองรับการถ่ายทอดสัญญาณแบบอินเตอร์รัปต์ ดังนั้นการติดต่อจากโฮสต์ไปยังอุปกรณ์จะต้องใช้การถ่ายทอดสัญญาณควบคุมเข้ามาช่วยแทน

4.13.2.2 โครงสร้างดิสคริปเตอร์

ไม่ว่าจะเป็นอุปกรณ์ USB ในคลาสหรือระดับใด ล้วนแล้วแต่ต้องมีดิสคริปเตอร์ในการทำงานทั้งสิ้น ใน HID คลาสเองก็เช่นกัน มีดิสคลิปเตอร์เป็นของตัวเองชื่อ HID ดิสคริปเตอร์ ซึ่งภายในดิสคริปเตอร์นี้จะแบ่งออกเป็น 2 ส่วนคือ ดิสคริปเตอร์รายงานผลการทำงาน หรือ รีพอร์ตดิสคริปเตอร์ (report descriptor) และกลุ่มดิสคลิปเตอร์ทางกายภาพ หรือ ฟิสิกัลดิสคริปเตอร์ (physical descriptor) รูปด้านล่างแสดงโครงสร้างดิสคริปเตอร์ของอุปกรณ์ USB ในระดับ HID จะเห็นได้ว่า HID ดิสคริปเตอร์จะบรรจุอยู่ในอินเตอร์เฟสดิสคริปเตอร์ร่วมกันเอ็นด์พอยต์ดิสคริปเตอร์

HID ดิสคริปเตอร์

- **Length** มีขนาด 1 ไบต์ ใช้ระบุขนาดรวมของดิสคริปเตอร์ในหน่วยไบต์ ใน HID คลาสจะกำหนดให้มีค่าเท่ากับ 09H
- **Descriptor type** มีขนาด 1 ไบต์ ใช้ระบุชนิดของดิสคริปเตอร์ ในกรณีเป็น HID ดิสคริปเตอร์จะมีค่าเท่ากับ 21H
- **HID vertion** มีขนาด 2 ไบต์ กำหนดในลักษณะรหัส BCD ใช้ระบุหมายเลขคุณสมบัติของ HID หรืออาจกล่าวได้ว่าเป็นการระบุเวอร์ชั่นก็ได้ โดยค่านี้จะใช้ตัวเลขฐานสิบ 4 ตัว 2 ตัวแรกใช้กำหนดตัวเลขเวอร์ชั่นหลัก ส่วน 2 ตัวหลังใช้กำหนดตัวเลขเวอร์ชั่นย่อย คั่นด้วยจุดทศนิยม ยกตัวอย่าง 0100 หมายถึง เวอร์ชั่น 1.0 ถ้าเป็นเวอร์ชั่น 1.1 ข้อมูลจะเป็น 0110 เป็นต้น
- **CountryCode** มีขนาด 1 ไบต์ใช้กำหนดรหัสของประเทศที่เกี่ยวข้องกับอุปกรณ์ USB ตัวนั้นๆ ขึ้น ตัวอย่างที่เห็นได้ชัดคือ คีย์บอร์ด เนื่องจากในแต่ละประเทศอาจมีการใช้คีย์บอร์ดแตกต่างกัน เช่น ในประเทศไทย, จีน,อังกฤษ เป็นต้น
- **NumDescriptors** มีขนาด 1 ไบต์ ใช้ระบุจำนวนดิสคลิปเตอร์ว่าเป็น รีพอร์ต หรือ ฟิสิกัลดิสคริปเตอร์ ถ้าเป็นรีพอร์ตดิสคริปเตอร์จะมีค่าเท่ากับ 22 *H*
- **DescriptorLength** มีขนาด 2 ไบต์ ใช้แจ้งขนาดหรือความยาวของรีพอร์ตดิสคริปเตอร์

4.13.2.3 รีพอร์ตดิสคริปเตอร์ (Report descriptor)

เป็นส่วนประกอบหลักที่อาจกล่าวได้ว่าสำคัญมากที่สุด เนื่องจากข้อมูลที่เก็บอยู่ในดิสคริปเตอร์ตัวนี้เป็นสิ่งที่อธิบายถึงรูปแบบและวิธีการใช้ข้อมูลเพื่อให้บรรลุวัตถุประสงค์ตามที่อุปกรณ์ USB ตัวนั้นกำหนดไว้ ยกตัวอย่าง หากอุปกรณ์ USB นี้เป็นเมาส์ ข้อมูลในรีพอร์ตดิสคริปเตอร์จะรายงานให้ทราบถึงการเคลื่อนที่ของเมาส์เพื่อระบุตำแหน่ง และสถานะการกำปุ่มของเมาส์ เป็นต้น

รีพอร์ตดิสคริปเตอร์จะมีความยาวเท่าใดก็ได้ แต่ต้องแจ้งความยาวลงใน DescriptorLength ภายใน HID ดิสคริปเตอร์ เพื่อให้โฮสต์ทราบด้วย หลักและเป้าหมายเบื้องต้นของการใช้งานรีพอร์ตดิสคริปเตอร์มีดังนี้

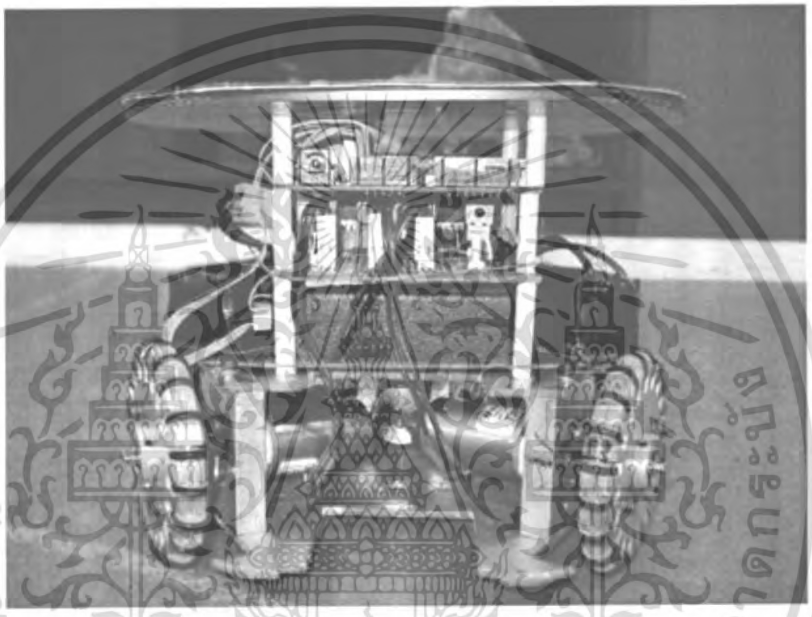
1. ใช้เก็บรายละเอียดข้อมูลของอุปกรณ์ให้ครบถ้วนที่สุดภายใต้พื้นที่ที่เล็กที่สุดเท่าที่เป็นไปได้
2. ยอมให้ซอฟต์แวร์ประยุกต์ที่เกี่ยวข้องสามารถข้ามข้อมูลที่ไม่ชัดเจนไปได้ เพื่อให้ยังคงสามารถดำเนินการต่อไปได้
3. สามารถขยายได้
4. รองรับการจัดเก็บและรวบรวมข้อมูล
5. สามารถอธิบายรายละเอียดด้วยข้อมูลภายในดิสคริปเตอร์เอง เพื่อประโยชน์ในการพัฒนาซอฟต์แวร์ประยุกต์สำหรับใช้งานร่วมกันได้

บทที่ 5

การออกแบบหุ่นยนต์เตะฟุตบอล

5.1 การติดตั้งโครงสร้างของหุ่นยนต์

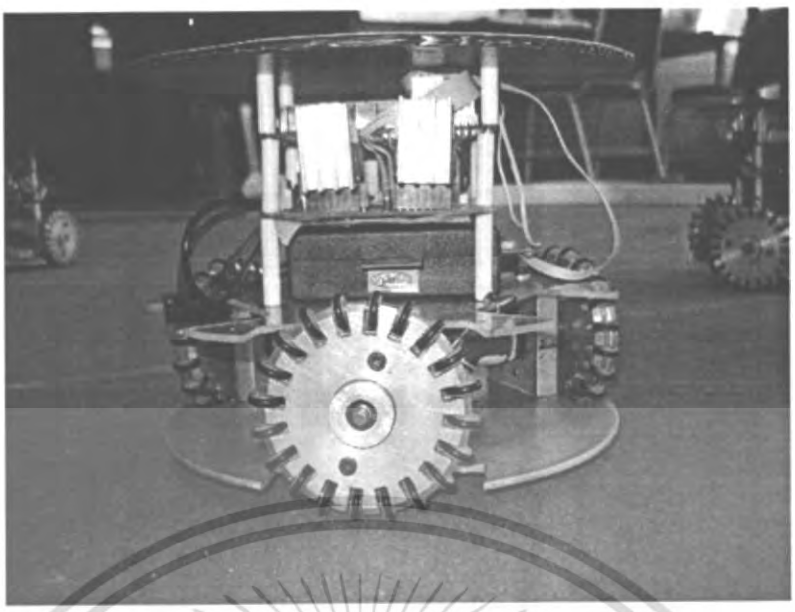
ส่วนประกอบของหุ่นยนต์จะใช้ล้อ OMNI DIRECTIONAL เพื่อให้หุ่นเคลื่อนที่ได้อิสระรอบทิศทาง ด้วยมอเตอร์ทดรอบ 12 Volt สามตัวติดตั้งในแนวนอนเพื่อลดพื้นที่ในการติดตั้งและมอเตอร์ขับล้อโดยตรง ลักษณะการติดตั้งดังภาพ



รูปที่ 5-1 แสดงการโครงสร้างของตัวหุ่นยนต์

5.2 การติดตั้งของล้อของตัวหุ่นยนต์

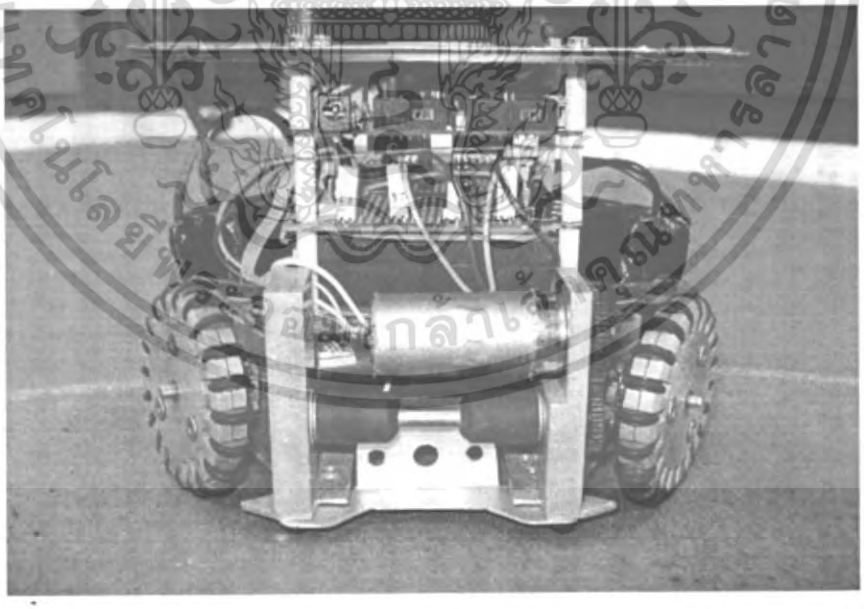
ล้อที่ใช้ในการติดตั้งจะเป็นล้อแบบ OMNI DIRECTIONAL ซึ่งจะเป็นที่สร้างขึ้นเองโดยเฉพาะ เพื่อให้การเคลื่อนที่ได้อิสระและจะใช้ตัวมอเตอร์ขับล้อโดยตรง โดยที่ไม่ต้องมีชุดเฟืองทดหรือชุดเฟืองเปลี่ยนทิศทาง และทำการติดตั้งแนวนอนเพื่อเป็นการลดความสูงของตัวหุ่นยนต์



รูปที่ 5-2 แสดงการติดตั้งตัวล้อขับของตัวหุ่นยนต์

5.3 การติดตั้งตัวเลี้ยงลูกบอลของตัวหุ่นยนต์

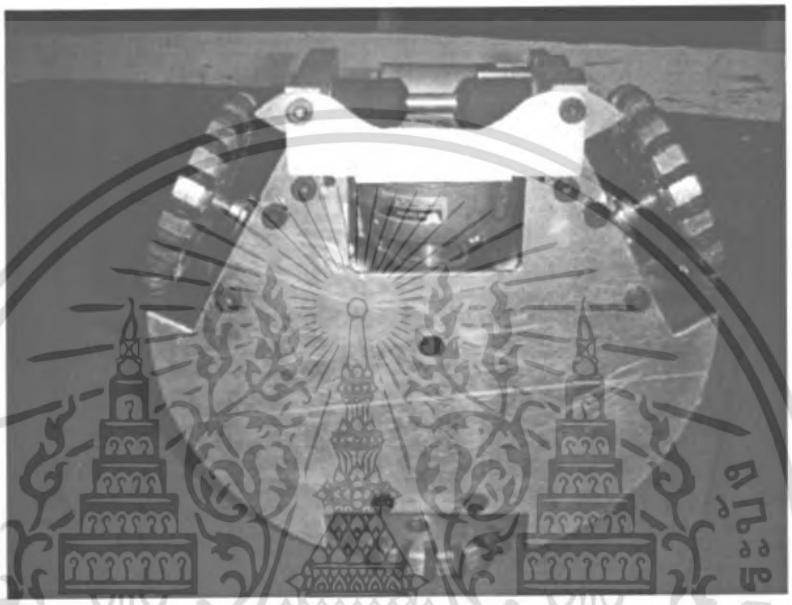
การติดตั้งตัวเลี้ยงบอลจะเป็นยางเพื่อให้เกิดความหนืดซึ่งเป็นสมบัติของยางในการปั่นบอลเข้ากับตัวหุ่นยนต์ เพื่อให้บอลติดกับตัวหุ่นยนต์ตลอดเวลาโดยปลายด้านหนึ่งของตัวชุดปั่นนี้จะต่อเฟืองส่งกำลังเข้ากับมอเตอร์ดังรูปที่ 5-3



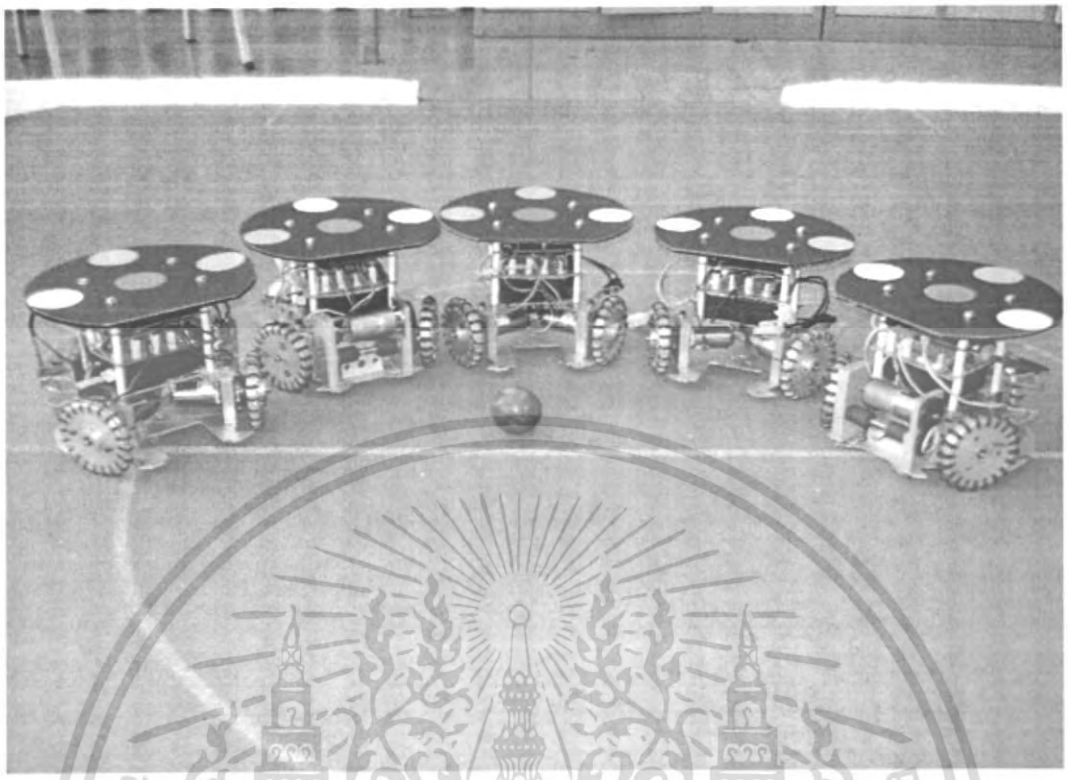
รูปที่ 5-3 แสดงการติดตั้งตัวเลี้ยงบอลของตัวหุ่นยนต์

5.4 การติดตั้งตัวยิงลูกบอลของตัวหุ่นยนต์

การติดตั้งตัวยิงลูกบอลถือเป็นอุปกรณ์อีกอย่างที่มีความสำคัญในการทำประตู ในส่วนอุปกรณ์หลักที่ใช้โซลินอยด์ 12 โวลต์ โดยตำแหน่งที่ติดตั้งนั้นจะอยู่ใต้ตัวเลี้ยงลูกบอลลงไป และตำแหน่งที่กระทบกับลูกบอลนั้นจะต้องโดนกึ่งกลางของลูกบอลพอดี เพื่อที่การยิงนั้นไม่ให้มีการหมุนย้อนกลับ ในส่วนตัวของโซลินอยด์จะต้องมีการติดตัวสปริงให้มีการดึงกลับหลังการยิงแล้วดังรูปที่ 5-4



รูปที่ 5-4 แสดงการติดตั้งตัวยิงลูกบอลของตัวหุ่นยนต์

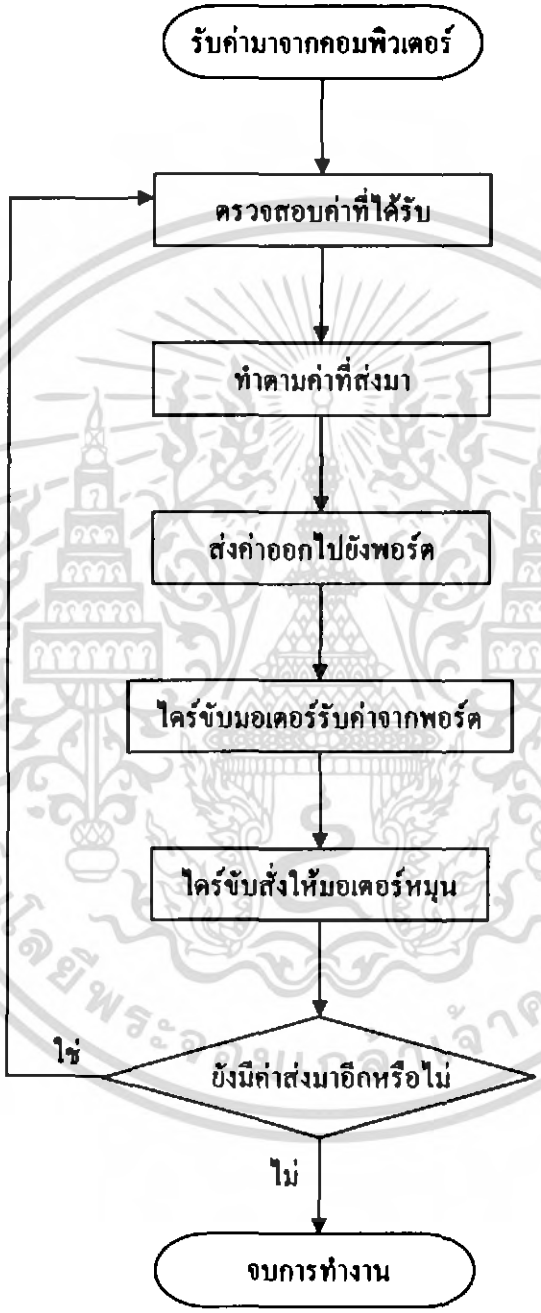


รูปที่ 5-5 หุ่นยนต์ครบทีม

บทที่ 6

ระบบควบคุมของหุ่นยนต์เตะฟุตบอล

6.1 การออกแบบโปรแกรมการสั่งงาน



รูปที่ 6-1 แผนผังลำดับการทำงานของโปรแกรมภาษาซีในไมโครคอนโทรลเลอร์

6.2 การแยกแยะตัวหุ่นยนต์

การแยกแยะหุ่นยนต์ (Robot Recognition)

ในการตรวจหาตำแหน่งของหุ่นยนต์นั้น ตามกติกาได้ให้ใช้สัญลักษณ์สีวงกลมติดไว้ที่ด้านบนตัวหุ่นยนต์ ซึ่งแต่ละทีมจะมีสีแตกต่างออกไป แต่การมีเพียงสัญลักษณ์เดียวทำให้ไม่สามารถบอกทิศทางของหุ่นยนต์ได้ เพราะในระบบควบคุมอาจจะมีความต้องการรู้ถึงทิศทางของหุ่นยนต์ โดยเฉพาะในกรณีที่หุ่นยนต์ติดระบบเลี้ยงและยิงลูกบอลไว้ด้านเดียว (ซึ่งโดยส่วนมากจะเป็นเช่นนี้) และในทีมหนึ่งก็ไม่ได้มีหุ่นยนต์เพียงตัวเดียว ทำให้ต้องมีการติดสัญลักษณ์สีเพิ่มเติมบนตัวหุ่นยนต์ การติดอาจมีรูปร่างแตกต่างกันไปทั้งสี่เหลี่ยม วงกลมก็สามารถใช้ได้

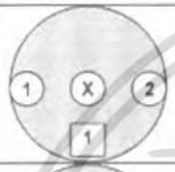
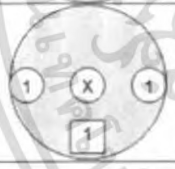
สำหรับโครงงานนี้ใช้สัญลักษณ์สีเพิ่มอีก 3 จุด จำนวน 2 สี ดังแสดงดังรูป ด้วยเหตุผลต่อไปนี้

รูปที่ 6-2 การติดสัญลักษณ์สีบนตัวหุ่นยนต์

1. การแยกแยะหุ่นยนต์ที่แข่งขันในสนามของทีมผู้ใช้จำนวน 5 ตัวหากใช้ 2 จุดจะไม่เพียงพอดังนั้นการใช้ 3 จุด จึงเพียงพอโดยจะแยกแยะหุ่นยนต์ได้สูงสุด 8 ตัวซึ่งเพียงพอ
2. โครงงานนี้เลือกใช้จำนวนจุดให้น้อยที่สุด เพื่อลดเวลาการประเมินผลและความผิดพลาดที่เกิดขึ้น
3. การใช้จุดสัญลักษณ์เพิ่มอีก 3 จุด รวมสัญลักษณ์ตามกติกาอีก 1 จุดเป็น 4 จุด และติดในรูปแบบดัง (รูปที่ 6-2) จะทำให้เหลือพื้นที่ด้านหนึ่งสำหรับใช้ประโยชน์อย่างอื่น เช่นช่องสำหรับควบคุมแบตเตอรี่ และที่สำคัญคือสามารถที่จะตัดออกให้เฉียงลง เพื่อให้สามารถเห็นลูกบอลที่อยู่ที่ชุดเลี้ยงชุดยิงได้ เพราะปัญหาที่พบบ่อยคือ เมื่อลูกบอลอยู่ใกล้หรือติดกับหุ่นยนต์ ด้วยความสูงของหุ่นยนต์จะทำให้ไปบดบังลูกบอลจนไม่สามารถตรวจจับได้

4. การติดสัญลักษณ์สีให้ห่างจากจุดกึ่งกลางให้มากที่สุด จะทำให้ความผิดพลาดในการคำนวณหาทิศทางของหุ่นยนต์ลดลง เพราะเนื่องจากหากอยู่ใกล้กันมาก ตำแหน่งที่ต่างไปเพียง1 Pixel ก็จะทำให้ค่าทิศทางเปลี่ยนไปมากขึ้น ดังนั้นการให้จุดสัญลักษณ์สีห่างกันมากขึ้น จะช่วยลดช่วงการแกว่งและความผิดพลาดของทิศทาง โดยการหาทิศทางจะใช้ตำแหน่งจริงของจุดทั้ง 4 มาคำนวณเพื่อหาทิศทางของหุ่นยนต์ด้านหน้า ว่าหันไปทิศทางใด

สำหรับรูปแบบการติดตั้งสัญลักษณ์สีดัง(รูปที่ 6-2) นั้นจะอ้างถึงหุ่นได้ทั้งหมด 8 แบบ แต่ในโครงการนี้จะใช้เพียง 5 แบบ โดยจะอ้างอิงถึงหุ่นหมายเลข 1 ถึง 5 ดังแสดงในรูปข้างล่างนี้

รูปแบบสัญลักษณ์สี	หุ่นยนต์หมายเลข	รูปแบบสัญลักษณ์สี	หุ่นยนต์หมายเลข
	1		2
	3		4
	5		6 (ไม่ได้ใช้)
	ไม่ได้ใช้		ไม่ได้ใช้

รูปที่ 6-3 รูปแบบการติดตั้งสัญลักษณ์

X = สีทีมของหุ่นตามกติกา (สีเหลืองหรือฟ้า)

1 = สีที่ 1 (Marker 1)

2 = สีที่ 2 (Marker 2)

บทที่ 7

การออกแบบโปรแกรมคำนวณพิกัดของตัวหุ่นยนต์เตะฟุตบอล

7.1 สาเหตุที่เลือกใช้โปรแกรมวิชวลเบสิกในการเขียนโปรแกรม

1. เป็นภาษาระดับกลาง มีลักษณะคล้ายภาษาอังกฤษ จึงทำให้ง่ายต่อการทำความเข้าใจและพัฒนาโปรแกรม

2. มีคำสั่งที่ใช้ในการติดต่อสื่อสารกับพอร์ตอนุกรมซึ่งสามารถนำไปควบคุมเครื่องได้

3. มีความสามารถในงานประเภทกราฟฟิคอยู่ในระดับที่น่าพอใจ ทำให้สามารถพัฒนาโปรแกรมได้อย่างสะดวก

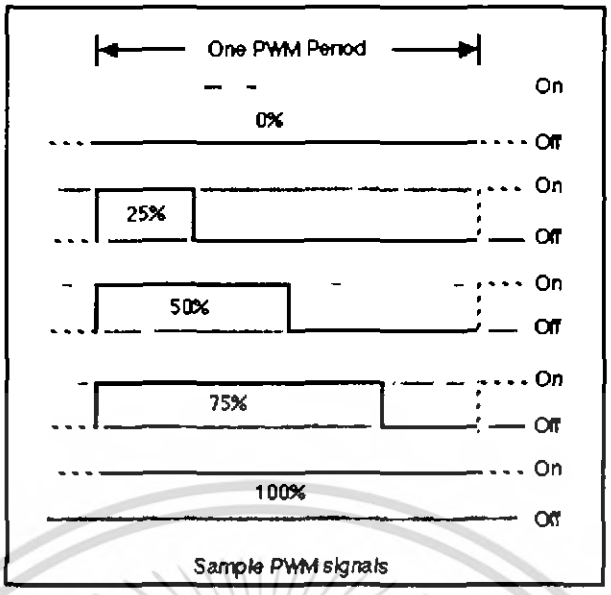
4. วิชวลเบสิกใช้พัฒนาโปรแกรมที่ใช้สื่อสารกับผู้ใช้เป็นหลัก ดังนั้นจึงสามารถพัฒนาหน้าตา (Interface) ของโปรแกรมออกมาได้สวยงามน่าใช้

7.2 การนำวิธีควบคุมหุ่นยนต์แบบ PWM (Pulse Width Modulation)

การควบคุมแบบ PWM คือ การควบคุมแรงดันให้กับโหลด (มอเตอร์ขับล้อ) เพื่อควบคุมความเร็ว โดยกำหนด Duty Cycle ของการ On และ Off เป็นเปอร์เซ็นต์

7.2.1 การทำงานของสัญญาณ PWM

- เส้นที่ 1 แสดงสัญญาณ PWM ที่ 0% duty cycle คือ สัญญาณในการออนจะเป็น 0% ของคาบสัญญาณ และ จะออฟเป็น 100% ของคาบสัญญาณ
- เส้นที่ 2 แสดงสัญญาณ PWM ที่ 25% duty cycle คือ สัญญาณในการออนจะเป็น 25% ของคาบสัญญาณ และ จะออฟเป็น 75% ของคาบสัญญาณ
- เส้นที่ 3 แสดงสัญญาณ PWM ที่ 50% duty cycle คือ สัญญาณในการออนจะเป็น 50% ของคาบสัญญาณ และ จะออฟเป็น 50% ของคาบสัญญาณ
- เส้นที่ 4 แสดงสัญญาณ PWM ที่ 75% duty cycle คือ สัญญาณในการออนจะเป็น 75% ของคาบสัญญาณ และ จะออฟเป็น 25% ของคาบสัญญาณ
- เส้นที่ 5 แสดงสัญญาณ PWM ที่ 100% duty cycle คือ สัญญาณในการออนจะเป็น 100% ของคาบสัญญาณ และ จะออฟเป็น 0% ของคาบสัญญาณ



รูปที่ 7-1 แสดงสัญญาณ PWM ซึ่งแสดงค่า duty cycles ที่ต่างๆกัน

7.2.2 เหตุผลที่ใช้ PWM ในการควบคุมความเร็วมอเตอร์

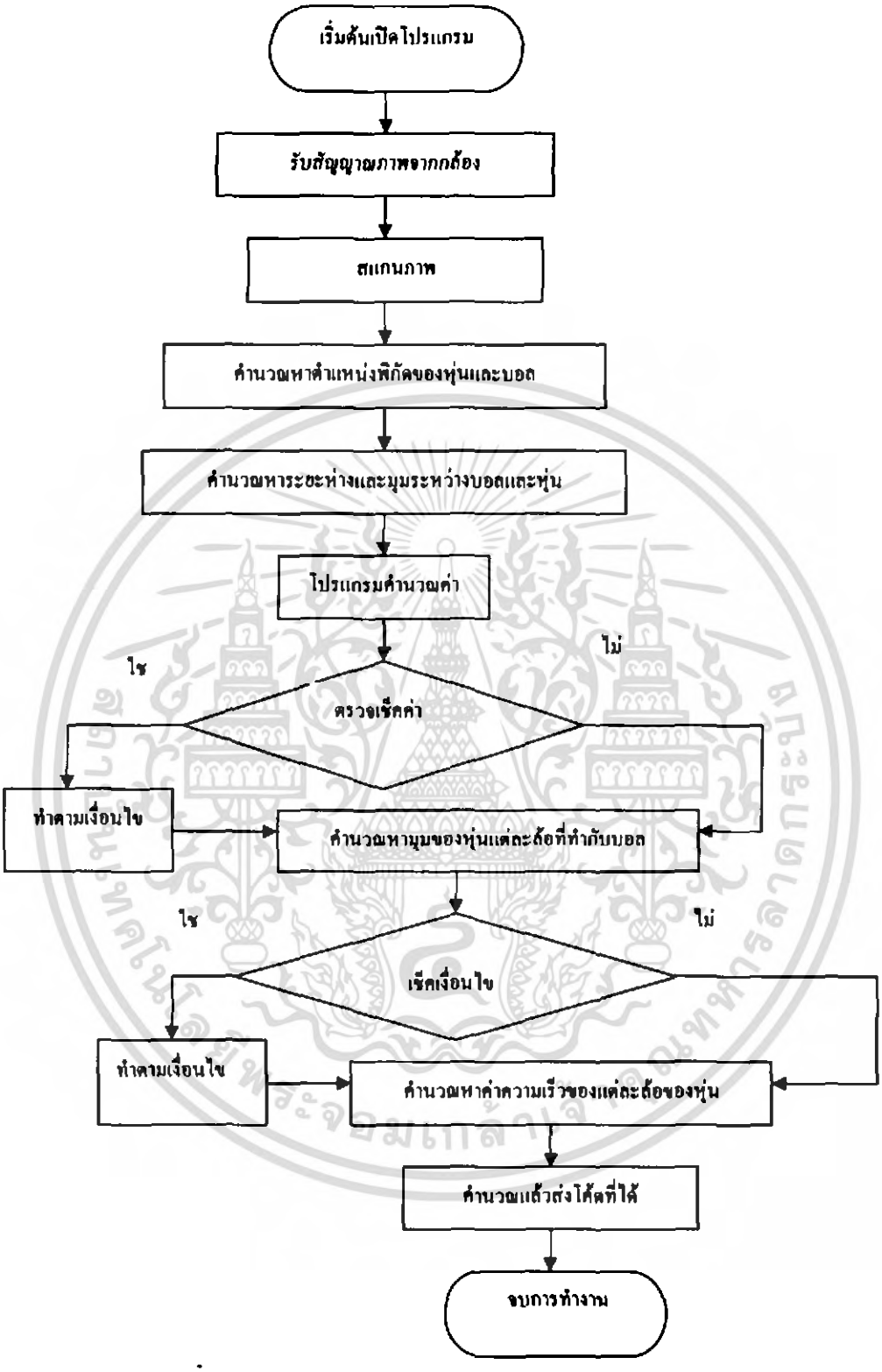
- PWM ง่ายในการอินเตอร์เฟสกับไมโครคอนโทรลเลอร์ และใช้เพียงแค่เอาท์พุตสัญญาณเดียวในการควบคุมความเร็ว
- PWM ทำหน้าที่บอกว่าจะจ่ายไฟเป็นกี่เปอร์เซ็นต์ของ duty cycle ในแต่มอเตอร์
- ไมโครคอนโทรลเลอร์ที่ใช้มีฟังก์ชั่นในการทำงานแบบ PWM โดยไมโครคอนโทรลเลอร์จะเปลี่ยนสัญญาณอินท์พุตที่รับเข้ามาให้เป็นสัญญาณเอาท์พุตแบบ PWM
- PWM ทำให้ได้ค่าความเร็วสูงสุดของมอเตอร์ เป็นเพราะ Power Supply จะจ่ายกำลังได้เต็มที่ทั้ง ON และ OFF(FULL ON and FULL OFF)
- เมื่อมีการควบคุมกำลังของแต่ละล้อจะทำให้หุ่นยนต์เคลื่อนที่ ในทิศทางต่างๆได้อย่างราบเรียบและแม่นยำยิ่งขึ้น

7.3 โปรแกรมคำนวณพิกัดในการเคลื่อนที่

การทำงานของโปรแกรมจะแบ่งขั้นตอนต่างๆออกเป็น 3 ส่วน คือส่วนการรับค่าจากกล้อง, ส่วนของการคำนวณค่ามุมของตัวหุ่นและระยะทางที่ทำกับลูกบอลและส่วนที่ทำการสร้างค่าของตัวอักษรที่ต้องการจะส่ง หลังจากนั้นก็ส่งออกไปที่พอร์ตอนุกรมได้เลยโดยมีการส่งออกมาที่ละตัวอักษรในระยะเวลาที่สามารถกำหนดได้ ไมโครคอนโทรลเลอร์ก็จะรับค่าที่ได้มาแปลงเป็นลักษณะการทำงานเพื่อสั่งให้ตัวหุ่นเคลื่อนที่ ในส่วนของการใช้งานพอร์ตยูเอสบีก็จะเขียนโปรแกรมติดต่อเหมือนกันแต่ว่าจะส่งค่าตัวแปรโดยตรงโดยไม่ต้องคำนวณหาค่าพิกัดที่ได้

ในขณะนี้การเคลื่อนที่ของหุ่นได้เริ่มมีการผสมผสานกันระหว่างการเลี้ยวและการเดินตรง ทำให้การเคลื่อนที่ของหุ่นมีความเร็วความราบเรียบและแม่นยำมากขึ้น ซึ่งเป็นผลมาจากการใช้วิธีควบคุมแบบ PWM ในการควบคุมหุ่น การเคลื่อนที่แบบนี้จะแตกต่างจากการเคลื่อนที่แบบเดิมคือ หุ่นจะเคลื่อนที่แบบแยกส่วนของการเลี้ยวและการเดินตรงออกจากกัน จึงทำให้การเคลื่อนที่เข้าหาลูกบอลเป็นไปด้วยความไม่รวดเร็ว

7.4 แผนผังลำดับการทำงานของโปรแกรม

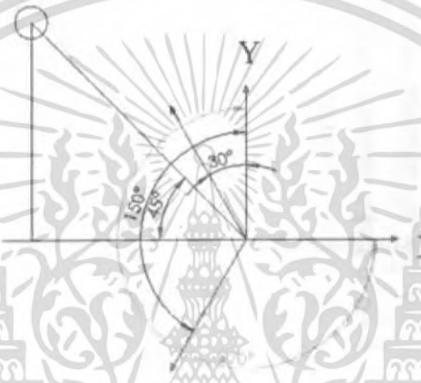


รูปที่ 7-2 แผนผังลำดับการทำงานของโปรแกรมวิชวลเบสิกในคอมพิวเตอร์

7.5 โปรแกรมการสร้างโค้ด (Generat Code Program)

จากการที่รูปภาพในคอมพิวเตอร์เกิดจากการนำจุดสีขนาดเล็กหรือที่เรียกกันว่า Pixels หลายๆจุดมาเรียงต่อกัน ประกอบกันเป็นภาพ เมื่อเราใช้ฟังก์ชันสแกนภาพที่ได้มาจากกล้องจะมีการแสดงค่าของสีซึ่งค่าสีในแต่ละจุดจะถูกจัดเก็บในระบบตัวเลข (Digital) และค่าของพิกัดในแกน x และ y ของตัวโปรแกรมเอง ซึ่งเราจะทำการเก็บค่าเฉพาะพิกัดมาคำนวณค่าของมุมและระยะทางที่เราต้องการได้

โปรแกรมคำนวณตำแหน่งจุดที่จะเคลื่อนที่จะอาศัยกฎของปีทากอรัสเปรียบเทียบค่าที่เราต้องการและนำเอาค่าความแตกต่างของมุมมาสร้างโค้ดที่ต้องการซึ่งจะทำให้หุ่นเคลื่อนที่ไปในตำแหน่งที่เราต้องการได้



รูปที่ 7-3 แสดงวิธีการคำนวณหามุมเพื่อนำค่าที่ได้ไปคำนวณหาค่าต่างๆ

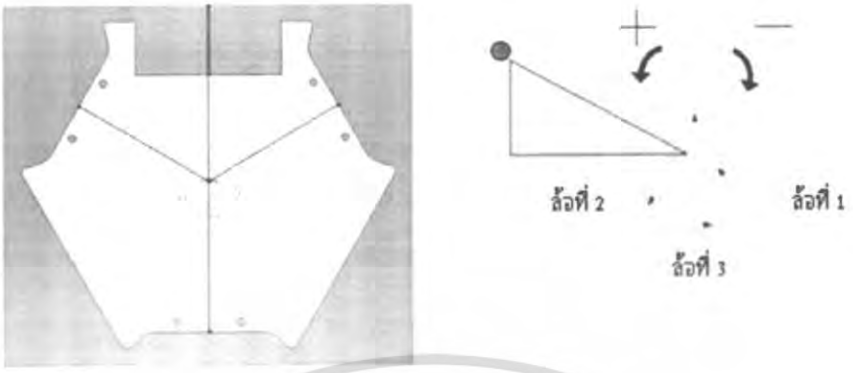
ตัวอย่างโค้ด

DegreeRobot = มุมหน้าหุ่น

Beta = มุมที่หุ่นทำกับลูกบอล

Zeta คือ มุมหน้าที่ทำกับลูกบอล

ตัวอย่างการคำนวณมุมของล้อ



รูปที่ 7-4 การวางตำแหน่งของล้อ

จาก มุมล้อที่ 1 = Zeta - 30
มุมล้อที่ 2 = Zeta - 150
มุมล้อที่ 3 = Zeta - 270

จาก(รูปที่ 7-3) และ (รูปที่ 7-4) ล้อทั้ง 3 จะทำมุมกันเท่ากับ 120 องศา และกำหนดให้ทิศทางการหมุนทวนเข็มนาฬิกาเป็นบวก และหมุนตามเข็มนาฬิกาเป็นลบ โดยมีจุดศูนย์กลางอยู่ที่ตัวหุ่นดังรูป ลูกศรที่ชี้ไปข้างหน้าทำมุมตั้งฉาก 90 องศา คือ มุมหน้าหุ่น

มุมที่ล้อ 1 ทำมุมเกินมุมหน้าหุ่นไป 30 องศา จึงเท่ากับมุมหน้าหุ่น ลบ มุมที่ล้อ 1 หรือมุมล้อที่ 1 = Zeta - 30

มุมที่ล้อ 2 ทำมุมเกินมุมหน้าหุ่นไป 150 องศา จึงเท่ากับมุมหน้าหุ่น ลบ มุมที่ล้อ 2 หรือมุมล้อที่ 2 = Zeta – 150

มุมที่ล้อ 3 ทำมุมเกินมุมหน้าหุ่นไป 270 องศา จึงเท่ากับมุมหน้าหุ่น ลบ มุมที่ล้อ 3หรือ มุมล้อที่ 3 = Zeta - 270

บทที่ 8

รูปแบบและผลการทดลอง

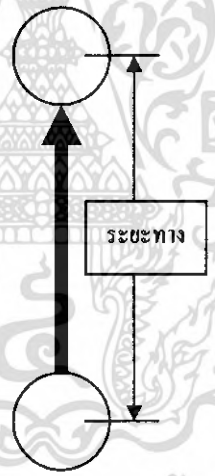
8.1 รูปแบบการทดลอง

รูปแบบของการทดลองเพื่อหาประสิทธิภาพรวมของเครื่องสร้างแบบสำหรับหุ่นยนต์เตะฟุตบอลนี้ คือ ส่วนของการเคลื่อนที่ของตัวหุ่นยนต์ที่เคลื่อนที่เป็นเส้นตรง

การทดลองนี้จะเป็นการตรวจเช็คว่าการทำงานของตัวหุ่นยนต์ยังมีข้อบกพร่องที่จะต้องแก้ไขบ้างหรือไม่ และจะทำการแก้ไขอย่างไรจึงจะทำให้การทำงานของตัวหุ่นยนต์ได้มีประสิทธิภาพมากที่สุด

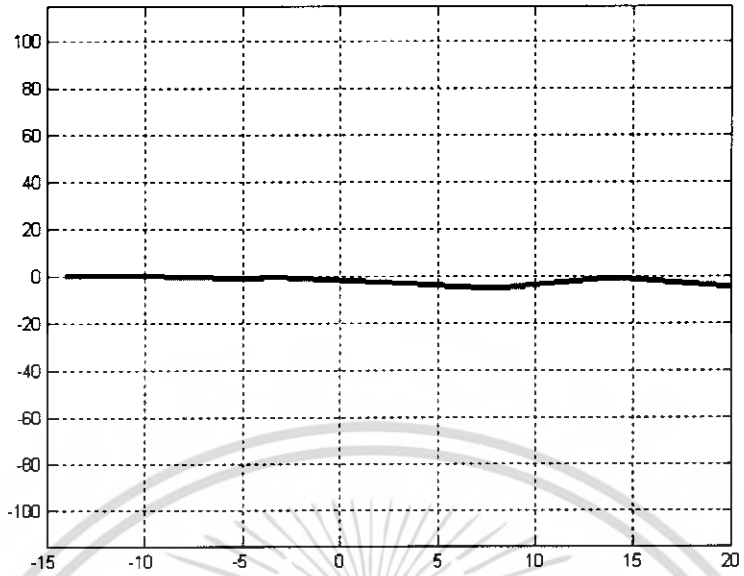
8.2 วิธีการทดลอง

จัดให้หุ่นยนต์หันเข้าจุดสีส้มและเปิดการทำงานร่วมกับกล้องวีดีโอ ในขณะเดียวกันนั้นจะมีโปรแกรมเก็บตำแหน่งการเคลื่อนที่ในแต่ละเฟรมที่กล้องส่งมาและนำค่าที่ได้ไปเขียนกราฟ ในการเขียนกราฟนั้นจะนำค่าที่ได้ในแต่ละเฟรมมาเทียบสเกลเพื่อคำนวณเป็นระยะทางแล้วนำมาเขียนกราฟ

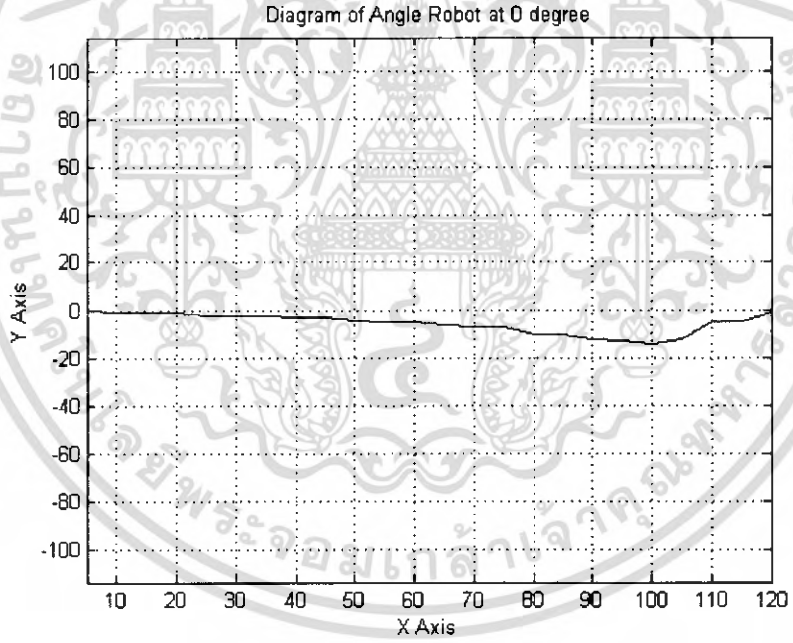


รูปที่ 8-1 แสดงถึงลักษณะการเคลื่อนที่แบบเส้นตรงที่ใช้ในการทดลอง

8.3 กราฟแสดงผลการทดลองการเคลื่อนที่ของหุ่นในทิศทางตรง



รูปที่ 8-2 แสดงเส้นทางการเคลื่อนที่ของหุ่นยนต์ครั้งใหม่



รูปที่ 8-3 แสดงเส้นทางการเคลื่อนที่ของหุ่นยนต์ ครั้งเก่า

จากการทดลองพบว่าเมื่อหุ่นเคลื่อนที่ไปได้ระยะหนึ่งหน้าหุ่นจะมีการเบี่ยงเบนออกจากเป้าหมายแต่หุ่นก็จะสามารถควบคุมตัวเองให้หันหน้าและเคลื่อนที่เข้าหาเป้าหมายได้อย่างดีขึ้น

บทที่ 9

สรุปผลการทดลองแนวทางแก้ไขและพัฒนา

9.1 สรุปผลการทดลอง

- หุ่นยนต์มีขนาดตามกติกาที่กำหนด
- หุ่นยนต์สามารถวิ่งเข้าหาบอลได้อย่างอัตโนมัติและเป็นทีม
- ในบางครั้งหุ่นอาจจะมีการเคลื่อนที่เบี่ยงเบนออกจากบอลไปบ้างแต่ก็สามารถวิ่งหันหน้ากลับเข้าหาบอลได้เพราะการทำงานของระบบเป็นระบบปิด
- บางครั้งหุ่นมีการหยุดชะงักขึ้นทำให้การเคลื่อนที่โดยรวมไม่ค่อยราบเรียบเป็นเพราะกล้องจับภาพไม่ได้ในชั่วขณะนั้น
- ในกรณีที่ปล่อยหุ่นลงสนามพร้อมกันหลายตัวยิ่งทำให้สัญญาณในการส่งจะช้าลง ทำให้การประมวลผลก็ช้าลงด้วย

9.2 แนวทางในการแก้ไขและพัฒนา

- เนื่องจากกล้องที่ใช้มีคุณภาพต่ำทำให้การอ่านสีจะมีความผิดพลาดอยู่เสมอ
- กล้องที่ใช้ควรจะมีอัตราเฟรม40-60 เฟรมต่อวินาที
- ห้องที่ใช้ทดสอบควรจะเป็นห้องทึบใช้แสงสว่างจากหลอดไฟฟ้าเพียงอย่างเดียวเพื่อให้แสงสว่างคงที่
- กล้องที่ใช้มีเพียงตัวเดียวทำให้ไม่สามารถมองเห็นได้ทั่วสนาม ดังนั้นจึงควรใช้กล้องไม่ต่ำกว่า 2 ตัวเพื่อที่จะมองเห็นได้ทั่วสนาม
- จำนวนคอมพิวเตอร์ที่ใช้อย่างน้อยต้องไม่ต่ำกว่าจำนวนกล้องที่ติดเพื่อที่การประมวลผลจะมีความรวดเร็วและแม่นยำยิ่งขึ้น
- ในส่วนของระบบควบคุมความเร็วล้อควรมีเอ็นโค้ดเดอร์ติดที่มอเตอร์เพื่อคอยตรวจสอบค่าที่ได้ว่าเป็นไปตามที่ต้องการหรือไม่
- ควรใช้ตัวรับส่งสัญญาณแบบตัวต่อตัวเพื่อป้องกันการสับสนของหุ่นและจะช่วยให้ประสิทธิภาพในการส่งสัญญาณดีขึ้น
- ชิ้นส่วนบางชิ้นเราสร้างด้วยมือทำให้ชิ้นส่วนอาจจะดูไม่สวยงามและไม่คงทนดังนั้นจึงควรสร้างชิ้นส่วนด้วยเครื่องจักรอัตโนมัติ เพื่อความสวยงามและคงทนซึ่งจะช่วยลดความคลาดเคลื่อนที่จะเกิดขึ้นได้ในระดับหนึ่ง
- เมื่อการเคลื่อนที่ของหุ่นสมบูรณ์แบบแล้วควรติดตั้งชุดเลี้ยงและชุดยิงเพื่อเพิ่มประสิทธิภาพของหุ่น
- ในการเขียนโปรแกรมควบคุมหุ่นยนต์ต้องอาศัยความรู้ประสบการณ์และความชำนาญของผู้เขียน ดังนั้นผู้เขียนโปรแกรมควบคุมหุ่นยนต์จะต้องศึกษาและทำความเข้าใจในตัวโปรแกรมมาเป็นอย่างดีและจะต้องเผื่อค่าผิดพลาดเข้าไปในตัวโปรแกรมด้วย

บรรณานุกรม

[1] อภิชาติ ภู่พลับ, "เริ่มต้นเขียนโปรแกรมติดต่อกันและควบคุมฮาร์ดแวร์ด้วย Visual Basic 6", Infopress Develoop Book, 2546

[2] ธนพล ฉันจรัสวิชัย, "ปฏิบัติการ Visual Basic สำหรับ Common Windows", ซีเอ็ดยูเคชั่น2546

[3] โชติพันธุ์ หล่อเลิศสุนทรและฐิตะพันธุ์ หล่อเลิศสุนทร, "สอนเขียน Visual Basic 6.0 ให้เป็นProject", Soft Express&Publishing, 2543

[4] สมศักดิ์ ศรีขจรเกียรติ, "Visual Basic 6. Teach Yourself", บิบลิโอไฟล์, 2542

[5] "Visual Basic Graphic Programming", John Wiley & sons, Inc, 1997

[6] A.F. Bakker, "Design of a high speed low friction XY-table", Phillips CentreFor industrial Technology (CFT)

[7] Hassian Z. Tameem, "Design and development of x-y positioning table using stepper motor and belt drive", Department of Mechanical and Production Engineering Yeshwantrao Chaven college of engineering Wanadongri,Nagpur

[8] วรพจน์ กรแก้ววัฒนกุล, ชัยวัฒน์ ลิ้มพรจิตรวิไล, "เรียนรู้และปฏิบัติการ ไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชฉบับ AT89C5X ของ Atmel," Innovative Experiment Co., Ltd.

[9] Joseph E. Shigley, Charles R. Mischke, Richard G. Budynas, "Mechanical Engineering Design",Seventh Edition,McGrawHill.

[10] www.thaiio.com

[11] www.pantip.com

[12] www.howstuffwork.com

[13] www.vbcode.com

[14] www.download.com

[15] www.google.co.th

ภาคผนวก ก

วิชวลเบสิกซอร์สโค้ด (Visual Basic Source Code)

```
********** VISION**********
Option Explicit

Private DataReceived() As VISIONDATA    ' Vision data received from clients
Private ReceivedCount As Integer    ' Number of data received from clients
Private sendDataTAI As Boolean



Private Sub chkRequest_Click()
   If (chkRequest.Value = vbChecked) Then
      cmdFind_Click
      'findai_Click
      tmrRequest.Enabled = True
      tmrReset.Enabled = True
      tmrFind.Enabled = True
   Else
      tmrReset.Enabled = False
      tmrRequest.Enabled = False
      tmrFind.Enabled = False
   End If
End Sub
Private Sub cmdFind_Click() ' Broadcast to find clients and find ai
      lstClient.Clear
      sockMain.RemoteHost = frmMain_BROADCAST
      sockMain.sendData "HELLO:" + sockMain.LocalIP
End Sub
Private Sub cmdMonitor_Click()
      frmMonitor.Show
```

```
    frmMonitor.Left = Me.Left
    frmMonitor.Top = Me.Top + Me.Height
    frmMonitor.rsMain.Redraw
End Sub
Private Sub cmdReset_Click()
    frmMonitor.rsMain.Reset
End Sub
Private Sub cmdSettings_Click()
    Load frmSettings
    frmSettings.Show vbModal
    ' Update
    tmrRuntime.Interval = frmMain_UPDATEDATA
    tmrRequest.Interval = frmMain_REQUEST
    tmrFind.Interval = frmMain_FINDCLIENT
    lblUnit.Caption = "(In " + UNIT_STRING + ")"
End Sub

Private Sub cmdStart_Click()

    Dim result As Long
    If (Text1.Text <> "") Then
      result = startConnection(Text1.hWnd)
    Else
      MsgBox "please enter ip"
    End If
      'Timer1.Enabled = True
      sendDataTAI = True

End Sub

Private Sub cmdStop_Click()
```

```
    Dim result As Long
    'Timer1.Enabled = False
    sendDataTAI = False
    result = closeConnection()

End Sub

Private Sub Command5_Click()
    Sent_Code = 5
    MSComm1.Output = Sent_Code
    MSComm1.PortOpen = False
End Sub

Private Sub findai_Click()
    'List1.Clear
    'Winsock1.RemoteHost = frmMain_BROADCAST
    'Winsock1.SendData "HIAI:" + Winsock1.LocalIP
End Sub
Private Sub Form_Load()
    'Init form & controls
    Text1.Text = ""
    sendDataTAI = False
    sockMain.RemotePort = frmMain_REMOTEPORT
    sockMain.RemoteHost = frmMain_BROADCAST
    sockMain.Bind frmMain_LOCALPORT
    'Winsock1.RemotePort = frmMain_AIREMPORT    'add
    'Winsock1.RemoteHost = frmMain_BROADCAST    'add
    'Winsock1.Bind frmMain_AIPORT               'add
    mnuLoad_Click   ' Load settings
    tmrRuntime.Interval = frmMain_UPDATEDATA
    tmrRequest.Interval = frmMain_REQUEST
```

```
    tmrFind.Interval = frmMain_FINDCLIENT
    lblUnit.Caption = "(In " + UNIT_STRING + ")"
    vsbFindClient_Change
    vsbRate_Change
    StartTime = Now
    cmdFind_Click   ' Find clients
    'findai_Click
End Sub
Private Sub Form_Unload(Cancel As Integer)
    sockMain.Close
    'Winsock1.Close 'add
    mnuSave_Click
    'End
End Sub
Public Sub UpdateData() ' Update data and controls
    'Write Code Doy na
    'Update data
    RunTime = Now - StartTime
    ' Update controls
    lblRunTime.Caption = "Running Time: " + Format$(Hour(RunTime), "0") + ":" +
Format$(Minute(RunTime), "00") + ":" + Format$(Second(RunTime), "00")
    lblReceive.Caption = "Updated: " + Format$(UpdateCount, "0")
    ' Update Robocup Status
      'robot 1
    Dim theta1 As Integer
    Dim theta2 As Integer
    Dim theta3 As Integer
    Dim velocity1 As Double
    Dim velocity2 As Double
    Dim velocity3 As Double
      'robot 2
```

```
Dim theta1_bot2 As Integer
Dim theta2_bot2 As Integer
Dim theta3_bot2 As Integer
Dim velocity1_bot2 As Double
Dim velocity2_bot2 As Double
Dim velocity3_bot2 As Double
  'robot 3
Dim theta1_bot3 As Integer
Dim theta2_bot3 As Integer
Dim theta3_bot3 As Integer
Dim velocity1_bot3 As Double
Dim velocity2_bot3 As Double
Dim velocity3_bot3 As Double
  'robot 4
Dim theta1_bot4 As Integer
Dim theta2_bot4 As Integer
Dim theta3_bot4 As Integer
Dim velocity1_bot4 As Double
Dim velocity2_bot4 As Double
Dim velocity3_bot4 As Double
  'robot 5
Dim theta1_bot5 As Integer
Dim theta2_bot5 As Integer
Dim theta3_bot5 As Integer
Dim velocity1_bot5 As Double
Dim velocity2_bot5 As Double
Dim velocity3_bot5 As Double
Dim i As Integer
Dim Robot As ROBOTDATA
Dim object As OBJECTDATA
Dim Degree_Ball As Integer
```

```
Dim Pos_Ball_X As Integer
Dim Pos_Ball_Y As Integer

Dim Pos_Robot_X1 As Integer
Dim Pos_Robot_Y1 As Integer
Dim Pos_Robot_X2 As Integer
Dim Pos_Robot_Y2 As Integer
Dim Pos_Robot_X3 As Integer
Dim Pos_Robot_Y3 As Integer
Dim Pos_Robot_X4 As Integer
Dim Pos_Robot_Y4 As Integer
Dim Pos_Robot_X5 As Integer
Dim Pos_Robot_Y5 As Integer
Dim del_x1 As Integer
Dim del_y1 As Integer
Dim del_x2 As Integer
Dim del_y2 As Integer
Dim del_x3 As Integer
Dim del_y3 As Integer
Dim del_x4 As Integer
Dim del_y4 As Integer
Dim del_x5 As Integer
Dim del_y5 As Integer
Dim Degree_Robot1 As Integer
Dim Degree_Robot2 As Integer
Dim Degree_Robot3 As Integer
Dim Degree_Robot4 As Integer
Dim Degree_Robot5 As Integer

Dim del_x As Integer
```

```
Dim del_y As Integer
Dim d1 As Integer
Dim d2 As Integer
Dim Degree_Robot As Integer
Dim deg1 As Integer
'Dim Alfa_bot2 As Double
'Dim Beta_bot2 As Double

Dim Beta1 As Double
Dim Beta2 As Double
Dim Beta3 As Double
Dim Beta4 As Double
Dim Beta5 As Double
Dim Zeta1 As Double
Dim Zeta2 As Double
Dim Zeta3 As Double
Dim Zeta4 As Double
Dim Zeta5 As Double
Dim Alfa As Double
Dim disp1 As Double
Dim Sent_Code As String
Dim Sent_Robot_A As String
Dim Sent_Robot_B As String
Dim Sent_Robot_C As String
Dim Sent_Robot_D As String
Dim Sent_Robot_E As String
Dim dis1 As Double
Dim dis2 As Double
Dim dis3 As Double
Dim dis4 As Double
Dim dis5 As Double
```

```
Dim aa As Integer
Dim delta_Robot_12 As Integer
aa = 20
MSComm1.Settings = "19200,n,8,1"
MSComm2.Settings = "19200,n,8,1"
If MSComm1.CommPort = 1 Then
  Else
   MSComm1.CommPort = 1
End If
If MSComm2.CommPort = 2 Then
  Else
   MSComm2.CommPort = 2
End If
  'MSComm1.OutBufferSize = 1
If MSComm1.PortOpen = True Then
  Else
   MSComm1.PortOpen = True
End If
If MSComm2.PortOpen = True Then
  Else
   MSComm2.PortOpen = True
End If
For i = 0 To 4
' Team's robots
frmMonitor.rsMain.GetRobot i, Robot.x, Robot.y, Robot.dir, Robot.valid
If (Robot.valid) Then
  lblRobot(i).Caption = "Robot " + Format$(i + 1, "0") + ": (" + Format$(Robot.x *
(frmMain_FIELD_MAXX), "0") + ", " + Format$(Robot.y * (frmMain_FIELD_MAXY), "0") + ") @ " +
Format$(Robot.dir * 180 / PI, "0")
  Else
    lblRobot(i).Caption = "Robot " + Format$(i + 1, "0") + ": " + frmMain_NATEXT
```

```
    End If
    ' Opponents
    frmMonitor.rsMain.GetOpponent i, object.x, object.y, object.valid
    If (object.valid) Then
      lblOpponent(i).Caption = "Opponent " + Format$(i + 1, "0") + ": (" + Format$(object.x *
(frmMain_FIELD_MAXX), "0") + ", " + Format$(object.y * (frmMain_FIELD_MAXY), "0") + ")"
      Else
        lblOpponent(i).Caption = "Opponent " + Format$(i + 1, "0") + ": " + frmMain_NATEXT
    End If

    If i = 0 Then
      'Degree Robot Use Compute
      Degree_Robot1 = Format$(Robot.dir * 180 / PI)
      Pos_Robot_Y1 = Format$(Robot.y * (frmMain_FIELD_MAXY))
      Pos_Robot_X1 = Format$(Robot.x * (frmMain_FIELD_MAXX))
    End If
    If i = 1 Then
      'Degree Robot Use Compute
      Degree_Robot2 = Format$(Robot.dir * 180 / PI)
      Pos_Robot_Y2 = Format$(Robot.y * (frmMain_FIELD_MAXY))
      Pos_Robot_X2 = Format$(Robot.x * (frmMain_FIELD_MAXX))
    End If
    If i = 2 Then
      'Degree Robot Use Compute
      Degree_Robot3 = Format$(Robot.dir * 180 / PI)
      Pos_Robot_Y3 = Format$(Robot.y * (frmMain_FIELD_MAXY))
      Pos_Robot_X3 = Format$(Robot.x * (frmMain_FIELD_MAXX))
    End If
    If i = 3 Then
      'Degree Robot Use Compute
      Degree_Robot4 = Format$(Robot.dir * 180 / PI)
```

```
        Pos_Robot_Y4 = Format$(Robot.y * (frmMain_FIELD_MAXY))
        Pos_Robot_X4 = Format$(Robot.x * (frmMain_FIELD_MAXX))
      End If
      If i = 4 Then
        'Degree Robot Use Compute
        Degree_Robot5 = Format$(Robot.dir * 180 / PI)
        Pos_Robot_Y5 = Format$(Robot.y * (frmMain_FIELD_MAXY))
        Pos_Robot_X5 = Format$(Robot.x * (frmMain_FIELD_MAXX))
      End If
    Next i
      lblBall1.Caption = "Ball1 " + ": (" + Format$(object.x * (frmMain_FIELD_MAXX), "0") + ", " +
Format$(object.y * (frmMain_FIELD_MAXY), "0") + ")"

      ' Ball
      frmMonitor.rsMain.GetBall object.x, object.y, object.valid
      If (object.valid) Then
        lblBall1.Caption = "Ball1 " + ": (" + Format$(object.x * (frmMain_FIELD_MAXX), "0") + ", " +
Format$(object.y * (frmMain_FIELD_MAXY), "0") + ")"
        lblBall2.Caption = "Ball2 " + ": (" + Format$(object.x * (frmMain_FIELD_MAXX), "I") + ", " +
Format$(object.y * (frmMain_FIELD_MAXY), "I") + ")"
        Else
          lblBall1.Caption = "Ball " + ": " + frmMain_NATEXT
          lblBall2.Caption = "Ball " + ": " + frmMain_NATEXT
      End If
      If 1 Then
        Pos_Ball_X = Format$(object.x * (frmMain_FIELD_MAXX))
        Pos_Ball_Y = Format$(object.y * (frmMain_FIELD_MAXY))
      End If
      'find Angle Robot
      del_y1 = Pos_Ball_Y - Pos_Robot_Y1
      del_x1 = Pos_Ball_X - Pos_Robot_X1
```

```
del_y2 = Pos_Ball_Y - Pos_Robot_Y2
del_x2 = Pos_Ball_X - Pos_Robot_X2
del_y3 = Pos_Ball_Y - Pos_Robot_Y3
del_x3 = Pos_Ball_X - Pos_Robot_X3
del_y4 = Pos_Ball_Y - Pos_Robot_Y4
del_x4 = Pos_Ball_X - Pos_Robot_X4
If Pos_Ball_Y > 20 And Pos_Ball_Y < 100 Then
   del_y5 = 25 - Pos_Robot_Y5
   del_x5 = -100 - Pos_Robot_X5
End If
If Pos_Ball_Y < -20 And Pos_Ball_Y > -100 Then
   del_y5 = -25 - Pos_Robot_Y5
   del_x5 = -100 - Pos_Robot_X5
End If
If Pos_Ball_Y > -20 And Pos_Ball_Y < 20 Then
   del_y5 = 0 - Pos_Robot_Y5
   del_x5 = -100 - Pos_Robot_X5
End If
   'Pos_Robot_Y1
If del_x1 = 0 Then
   del_x1 = 0.01
End If
If del_y1 = 0 Then
   del_y1 = 0.01
End If
If del_x2 = 0 Then
   del_x2 = 0.01
End If
If del_y2 = 0 Then
   del_y2 = 0.01
End If
```

```
If del_x3 = 0 Then
   del_x3 = 0.01
End If
If del_y3 = 0 Then
   del_y3 = 0.01
End If
If del_x4 = 0 Then
   del_x4 = 0.01
End If
If del_y4 = 0 Then
   del_y4 = 0.01
End If
If del_x5 = 0 Then
   del_x5 = 0.01
End If
If del_y5 = 0 Then
   del_y5 = 0.01
End If
dis1 = Sqr((del_x1 / aa * del_x1 / aa + del_y1 / aa * del_y1 / aa))
dis2 = Sqr((del_x2 / aa * del_x2 / aa + del_y2 / aa * del_y2 / aa))
dis3 = Sqr((del_x3 / aa * del_x3 / aa + del_y3 / aa * del_y3 / aa))
dis4 = Sqr((del_x4 / aa * del_x4 / aa + del_y4 / aa * del_y4 / aa))
dis5 = Sqr((del_x5 / aa * del_x5 / aa + del_y5 / aa * del_y5 / aa))
Command16.Caption = dis1
Command17.Caption = dis2
If del_x1 = 0 Then
   del_x1 = 1
End If
If del_x2 = 0 Then
   del_x2 = I
End If
```

```
If del_x3 = 0 Then
   del_x3 = 1
End If
If del_x4 = 0 Then
   del_x4 = 1
End If
If del_x5 = 0 Then
   del_x5 = 1
End If

Beta1 = Atn(del_y1 / del_x1) * 180 / PI \ 1
If (del_y1 < 0 And del_x1 < 0) Or (del_y1 > 0 And del_x1 < 0) Then
   Beta1 = Beta1 + 180
End If
Beta2 = Atn(del_y2 / del_x2) * 180 / PI \ 1
If (del_y2 < 0 And del_x2 < 0) Or (del_y2 > 0 And del_x2 < 0) Then
   Beta2 = Beta2 + 180
End If
Beta3 = Atn(del_y3 / del_x3) * 180 / PI \ 1
If (del_y3 < 0 And del_x3 < 0) Or (del_y3 > 0 And del_x3 < 0) Then
   Beta3 = Beta3 + 180
End If
Beta4 = Atn(del_y4 / del_x4) * 180 / PI \ 1
If (del_y4 < 0 And del_x4 < 0) Or (del_y4 > 0 And del_x4 < 0) Then
   Beta4 = Beta4 + 180
End If
Beta5 = Atn(del_y5 / del_x5) * 180 / PI \ 1
If (del_y5 < 0 And del_x5 < 0) Or (del_y5 > 0 And del_x5 < 0) Then
   Beta5 = Beta5 + 180
End If
Command12.Caption = Beta1 & "  " & Beta2 & "  " & Beta3 & "  " & Beta4 & "  " & Beta5
```

```
'lblBall.Caption = lblBall.Caption + "@" & Beta
'del_y2 = Pos_Ball_Y - Pos_Robot_Y2
'del_x2 = Pos_Ball_X - Pos_Robot_X2
'd1 = Sqr((del_x1 * del_x1) + (del_y1 * del_y1))
'd2 = Sqr((del_x2 * del_x2) + (del_y2 * del_y2))
'Beta = Atn(Abs(a)) * 180 / PI \ 1
'lblBall.Caption = lblBall.Caption + "@" & Beta
'lblBall.Caption = lblBall.Caption + "@" & Beta1

Zeta1 = Beta1 - Degree_Robot1
Zeta2 = Beta2 - Degree_Robot2
Zeta3 = Beta3 - Degree_Robot3
Zeta4 = Beta4 - Degree_Robot4
Zeta5 = Beta5 - Degree_Robot5

If Zeta1 < -180 Then
   Zeta1 = 360 + Zeta1
End If
If Zeta1 > 180 Then
   Zeta1 = Zeta1 - 360
End If
If Zeta2 < -180 Then
   Zeta2 = 360 + Zeta2
End If
If Zeta2 > 180 Then
   Zeta2 = Zeta2 - 360
End If
If Zeta3 < -180 Then
   Zeta3 = 360 + Zeta3
End If
If Zeta3 > 180 Then
```

```
    Zeta3 = Zeta3 - 360
End If
If Zeta4 < -180 Then
    Zeta4 = 360 + Zeta4
End If
If Zeta4 > 180 Then
    Zeta4 = Zeta4 - 360
End If
If Zeta5 < -180 Then
    Zeta5 = 360 + Zeta5
End If
If Zeta5 > 180 Then
    Zeta5 = Zeta5 - 360
End If
Command11.Caption = Zeta1 & "  " & Zeta2 & "  " & Zeta3 & "  " & Zeta4 & "  " & Zeta5
Command1.Caption = Zeta1
Command2.Caption = Alfa
'Angle robot 3
theta1 = Zeta1 - 30
theta2 = Zeta1 - 150
theta3 = Zeta1 - 270
theta1_bot2 = Zeta2 - 30
theta2_bot2 = Zeta2 - 150
theta3_bot2 = Zeta2 - 270
theta1_bot3 = Zeta3 - 30
theta2_bot3 = Zeta3 - 150
theta3_bot3 = Zeta3 - 270
theta1_bot4 = Zeta4 - 30
theta2_bot4 = Zeta4 - 150
theta3_bot4 = Zeta4 - 270
theta1_bot5 = Zeta5 - 30
```

```
theta2_bot5 = Zeta5 - 150
theta3_bot5 = Zeta5 - 270

'computer PWM  max velocity = 1
velocity1 = (Cos(theta1 * PI / 180) * 150) \ 1
velocity2 = (Cos(theta2 * PI / 180) * 150) \ 1
velocity3 = (Cos(theta3 * PI / 180) * 150) \ 1
velocity1_bot2 = (Cos(theta1_bot2 * PI / 180) * 200) \ 1
velocity2_bot2 = (Cos(theta2_bot2 * PI / 180) * 200) \ 1
velocity3_bot2 = (Cos(theta3_bot2 * PI / 180) * 200) \ 1
velocity1_bot3 = (Cos(theta1_bot3 * PI / 180) * 200) \ 1
velocity2_bot3 = (Cos(theta2_bot3 * PI / 180) * 200) \ 1
velocity3_bot3 = (Cos(theta3_bot3 * PI / 180) * 200) \ I
velocity1_bot4 = (Cos(theta1_bot4 * PI / 180) * 200) \ 1
velocity2_bot4 = (Cos(theta2_bot4 * PI / 180) * 200) \ 1
velocity3_bot4 = (Cos(theta3_bot4 * PI / 180) * 200) \ 1
velocity1_bot5 = (Cos(theta1_bot5 * PI / 180) * 200) \ 1
velocity2_bot5 = (Cos(theta2_bot5 * PI / 180) * 200) \ 1
velocity3_bot5 = (Cos(theta3_bot5 * PI / 180) * 100) \ 1

Command8.Caption = velocity1
Command9.Caption = velocity2
Command10.Caption = velocity3
Command13.Caption = velocity1_bot2
Command14.Caption = velocity2_bot2
Command15.Caption = velocity3_bot2
'Robot 1 Conput Position and angle moving
If dis1 < dis2 And dis1 < dis3 And dis1 < dis4 Then
   If Zeta1 > 0 And Zeta1 < 60 Then
      Sent_Robot_A = "A" + Chr(Abs(velocity1)) + "0" + "0" + Chr(Abs(velocity2)) +
Chr(Abs(velocity3)) + "0" + "zzzz"
```

```
        End If
        If Zeta1 > 60 And Zeta1 < 180 Then
           Sent_Robot_A = "A" + Chr(100) + "0" + Chr(100) + "0" + Chr(100) + Chr(0) + "zzzz"
        End If
        If Zeta1 > -60 And Zeta1 < 0 Then
           Sent_Robot_A = "A" + Chr(Abs(velocity1)) + "0" + "0" + Chr(Abs(velocity2)) + "0" +
Chr(Abs(velocity3)) + "zzzz"
        End If
        If Zeta1 > -180 And Zeta1 < -60 Then
           Sent_Robot_A = "A" + "0" + Chr(100) + "0" + Chr(100) + "0" + Chr(100) + "zzzz"
        End If
        MSComm1.Output = Sent_Robot_A + "B00000000" + "C00000000" + "D0000000"

     End If

     'Robot 2 Conput Position and angle moving
     If dis2 < dis1 And dis2 < dis3 And dis2 < dis4 Then
        If Zeta2 > 0 And Zeta2 < 60 Then
           Sent_Robot_B = "B" + Chr(Abs(velocity1_bot2)) + "0" + "0" + Chr(Abs(velocity2_bot2)) +
Chr(Abs(velocity3_bot2)) + "0" + "zzzz"
        End If
        If Zeta2 > 60 And Zeta2 < 180 Then
           Sent_Robot_B = "B" + Chr(150) + "0" + Chr(150) + "0" + Chr(150) + Chr(0) + "zzzz"
        End If
        If Zeta2 > -60 And Zeta2 < 0 Then
           Sent_Robot_B = "B" + Chr(Abs(velocity1_bot2)) + "0" + "0" + Chr(Abs(velocity2_bot2)) + "0" +
Chr(Abs(velocity3_bot2)) + "zzzz"
        End If
        If Zeta2 > -180 And Zeta2 < -60 Then
           Sent_Robot_B = "B" + "0" + Chr(100) + "0" + Chr(100) + "0" + Chr(100) + "zzzz"
        End If
```

```
        MSComm1.Output = Sent_Robot_B + "A00000000" + "C00000000" + "D0000000"
    End If

    'Robot 3 Conput Position and angle moving
    If dis3 < dis1 And dis3 < dis2 And dis3 < dis4 Then
        If Zeta3 > 0 And Zeta3 < 60 Then
            Sent_Robot_C = "C" + Chr(Abs(velocity1_bot3)) + "0" + "0" + Chr(Abs(velocity2_bot3)) +
Chr(Abs(velocity3_bot3)) + "0" + "zzzz"
        End If
        If Zeta3 > 60 And Zeta3 < 180 Then
            Sent_Robot_C = "C" + Chr(150) + "0" + Chr(150) + "0" + Chr(150) + Chr(0) + "zzzz"
        End If
        If Zeta3 > -60 And Zeta3 < 0 Then
            Sent_Robot_C = "C" + Chr(Abs(velocity1_bot3)) + "0" + "0" + Chr(Abs(velocity2_bot3)) + "0" +
Chr(Abs(velocity3_bot3)) + "zzzz"
        End If
        If Zeta3 > -180 And Zeta3 < -60 Then
            Sent_Robot_C = "C" + "0" + Chr(100) + "0" + Chr(100) + "0" + Chr(100) + "zzzz"
        End If
        MSComm1.Output = Sent_Robot_C + "A00000000" + "B0000000" + "D0000000"
    End If

    'Robot 4 Conput Position and angle moving
    If dis4 < dis1 And dis4 < dis2 And dis4 < dis3 Then
        If Zeta4 > 0 And Zeta4 < 60 Then
            Sent_Robot_D = "D" + Chr(Abs(velocity1_bot4)) + "0" + "0" + Chr(Abs(velocity2_bot4)) +
Chr(Abs(velocity3_bot4)) + "0" + "zzzz"
        End If
        If Zeta4 > 60 And Zeta4 < 180 Then
            Sent_Robot_D = "D" + Chr(150) + "0" + Chr(150) + "0" + Chr(150) + Chr(0) + "zzzz"
        End If
```

```
        If Zeta4 > -60 And Zeta4 < 0 Then
            Sent_Robot_D = "D" + Chr(Abs(velocity1_bot4)) + "0" + "0" + Chr(Abs(velocity2_bot4)) + "0" +
Chr(Abs(velocity3_bot4)) + "zzzz"
        End If
        If Zeta4 > -180 And Zeta4 < -60 Then
            Sent_Robot_D = "D" + "0" + Chr(100) + "0" + Chr(100) + "0" + Chr(100) + "zzzz"
        End If
        MSComm1.Output = Sent_Robot_D + "A00000000" + "B0000000" + "C0000000"
    End If

    'Robot 5 Conput Position and angle moving
    If 1 Then
        If Zeta5 > 0 And Zeta5 < 60 Then
            Sent_Robot_E = "E" + Chr(Abs(velocity1_bot5)) + "0" + "0" + Chr(Abs(velocity2_bot5)) +
Chr(Abs(velocity3_bot5)) + "0" + "zzzz"
        End If
        If Zeta5 > 60 And Zeta5 < 180 Then
            Sent_Robot_E = "E" + Chr(100) + "0" + Chr(100) + "0" + Chr(100) + Chr(0) + "zzzz"
        End If
        If Zeta5 > -60 And Zeta5 < 0 Then
            Sent_Robot_E = "E" + Chr(Abs(velocity1_bot5)) + "0" + "0" + Chr(Abs(velocity2_bot5)) + "0" +
Chr(Abs(velocity3_bot5)) + "zzzz"
        End If
        If Zeta5 > -180 And Zeta5 < -60 Then
            Sent_Robot_E = "E" + "0" + Chr(100) + "0" + Chr(100) + "0" + Chr(100) + "zzzz"
        End If
        MSComm1.Output = Sent_Robot_E

    End If
End Sub
```

```
Private Sub mnuAbout_Click()
    frmAbout.Show vbModal
End Sub
Private Sub mnuExit_Click()
    Unload Me
End Sub
Private Sub mnuLoad_Click()
    LoadSettings
End Sub
Private Sub mnuSave_Click()
    SaveSettings
End Sub

Private Sub sockMain_DataArrival(ByVal bytesTotal As Long)
    If (bytesTotal <= 1) Then Exit Sub
    Dim Data As String
    Dim DataArr() As String
    Dim i As Integer, Packet As Long
    Dim FromIP As String
    Dim tmpflag As Boolean
    Dim o_ject As OBJECTDATA
    Dim R_bot As ROBOTDATA

    sockMain.GetData Data, vbString
    DataArr = Split(Data, ":")
    Select Case DataArr(0)
      Case "HELLO":   ' Reply from client
        FromIP = DataArr(1)
        tmpflag = True  ' Assume add
        ' Check existing, if not found then add
        For i = 0 To (lstClient.ListCount - 1)
```

```
            If (lstClient.List(i) = FromIP) Then
              tmpflag = False  ' Don't add
            Exit For
          End If
        Next i
          If (tmpflag) Then
            If (lstClient.ListCount <= frmMain_MAXCLIENT) Then  ' Limit number of client
              lstClient.AddItem DataArr(1)    ' Add client to list
              ReDim DataReceived(0 To (lstClient.ListCount - 1)) As VISIONDATA
            End If
          End If
          Case "DATA":    ' Receive data from a client, update them
            FromIP = DataArr(1)
            Packet = 2
          If (ReceivedCount < lstClient.ListCount) Then
            If (DataArr(2) = "T") Then
              DataReceived(ReceivedCount).Ball.valid = True
              DataReceived(ReceivedCount).Ball.x = Val(DataArr(3))
              DataReceived(ReceivedCount).Ball.y = Val(DataArr(4))
              Packet = Packet + 3
            Else
              'DataReceived(ReceivedCount).Ball.valid = False
              Packet = Packet + 1
            End If
            For i = 0 To 4
            If (DataArr(Packet) = "T") Then
              DataReceived(ReceivedCount).Opponent(i).valid = True
              DataReceived(ReceivedCount).Opponent(i).x = Val(DataArr(Packet + 1))
              DataReceived(ReceivedCount).Opponent(i).y = Val(DataArr(Packet + 2))
              Packet = Packet + 3
            Else
```

```
            'DataReceived(ReceivedCount).Opponent(i).valid = False
            Packet = Packet + 1
        End If
        Next i
        For i = 0 To 4
        If (DataArr(Packet) = "T") Then
            DataReceived(ReceivedCount).Robot(i).valid = True
            DataReceived(ReceivedCount).Robot(i).x = Val(DataArr(Packet + 1))
            DataReceived(ReceivedCount).Robot(i).y = Val(DataArr(Packet + 2))
            DataReceived(ReceivedCount).Robot(i).dir = Val(DataArr(Packet + 3))
            Packet = Packet + 4
        Else
            'DataReceived(ReceivedCount).Robot(i).valid = False
            Packet = Packet + 1
        End If
        Next i
        ReceivedCount = ReceivedCount + 1
    End If

' Update Data (Average them)
If (ReceivedCount >= lstClient.ListCount) And (lstClient.ListCount > 0) Then
    Dim sumX As Double, sumY As Double
    Dim sumSine As Double, sumCosine As Double
    Dim Count As Integer
    Dim addnew As Boolean
    Dim j As Integer, k As Integer
    Dim OpponentX(0 To 4) As New clsAvrNumber
    Dim OpponentY(0 To 4) As New clsAvrNumber
    Dim OppCount As Integer
    Dim result As Long
```

```
        ' Ball
        sumX = 0: sumY = 0
        Count = 0
        For i = 0 To (ReceivedCount - 1)
          If (DataReceived(i).Ball.valid) Then
            sumX = sumX + DataReceived(i).Ball.x
            sumY = sumY + DataReceived(i).Ball.y
            Count = Count + 1
          End If
        Next i
        If (Count > 0) Then
          frmMonitor.rsMain.SetBall sumX / Count / frmMain_FIELD_MAXX, sumY / Count /
frmMain_FIELD_MAXY, True

          If (sendDataTAI = True) Then
            frmMonitor.rsMain.GetBall o_ject.x, o_ject.y, o_ject.valid
            'tmpstr = tmpstr + ":T:" + Format$(ob_ject.X * (frmMain_FIELD_MAXX), "0") + ":" +
Format$(ob_ject.Y * (frmMain_FIELD_MAXY), "0")
            result = sendData(Ball, 0, CLng(o_ject.x * (frmMain_FIELD_MAXX)), CLng(o_ject.y *
(frmMain_FIELD_MAXY)), 360)
          End If

        Else
          frmMonitor.rsMain.SetBall 0, 0, False
        End If

        ' Opponents (Match them)
        OppCount = 0
        For i = 0 To 4
          OpponentX(i).CreateArray frmMain_MAXCLIENT
          OpponentY(i).CreateArray frmMain_MAXCLIENT
```

```
        Next i
        For i = 0 To (ReceivedCount - 1)
          For j = 0 To 4
            If (DataReceived(i).Opponent(j).valid) Then
              addnew = True
              For k = 0 To (OppCount - 1) ' Find the same point
                If (Distance(DataReceived(i).Opponent(j).x, DataReceived(i).Opponent(j).y,
OpponentX(k).GetAverage, OpponentY(k).GetAverage) <= frmMain_SAMEPOINT) Then
                  ' Add value to existing
                    OpponentX(k).AddValue DataReceived(i).Opponent(j).x
                    OpponentY(k).AddValue DataReceived(i).Opponent(j).y
                    addnew = False
                  Exit For
                End If
              Next k
              If (addnew) Then
                If (OppCount < 5) Then
                ' Add new opponent
                  OpponentX(OppCount).AddValue DataReceived(i).Opponent(j).x
                  OpponentY(OppCount).AddValue DataReceived(i).Opponent(j).y
                  OppCount = OppCount + 1
                End If
              End If
            End If
          Next j
        Next i
        ' Update opponent data
        For j = 0 To (OppCount - 1)
          frmMonitor.rsMain.SetOpponent j, OpponentX(j).GetAverage / frmMain_FIELD_MAXX,
OpponentY(j).GetAverage / frmMain_FIELD_MAXY, True
```

```
        If (sendDataTAI = True) Then

            frmMonitor.rsMain.GetOpponent j, o_ject.x, o_ject.y, o_ject.valid
            'tmpstr = tmpstr + ":T:" + Format$(ob_ject.X * (frmMain_FIELD_MAXX), "0") + ":" +
Format$(ob_ject.Y * (frmMain_FIELD_MAXY), "0")

            result = sendData(DEVIL, j, CLng(o_ject.x * (frmMain_FIELD_MAXX)), CLng(o_ject.y *
(frmMain_FIELD_MAXY)), 360)
        End If
      Next j
      For j = OppCount To 4 ' Set other opponents to invalid
        frmMonitor.rsMain.SetOpponent j, 0, 0, False
      Next j

      ' Our Team Robots R_bot
      For j = 0 To 4
        sumX = 0: sumY = 0
        Count = 0
        sumSine = 0: sumCosine = 0
        For i = 0 To (ReceivedCount - 1)
          If (DataReceived(i).Robot(j).valid) Then
            sumX = sumX + DataReceived(i).Robot(j).x
            sumY = sumY + DataReceived(i).Robot(j).y
              sumSine = sumSine + Sin(DataReceived(i).Robot(j).dir * PI / 180)
            sumCosine = sumCosine + Cos(DataReceived(i).Robot(j).dir * PI / 180)
            Count = Count + 1
          End If
        Next i
        If (Count > 0) Then
          Dim radian As Double
          radian = Atn((sumSine / Count) / (sumCosine / Count))
```

```
            If (sumCosine < 0) Then radian = radian + PI
            If (radian < 0) Then radian = radian + (2 * PI)
            frmMonitor.rsMain.SetRobot j, sumX / Count / frmMain_FIELD_MAXX, sumY / Count /
frmMain_FIELD_MAXY, radian, True

            If (sendDataTAI = True) Then

              frmMonitor.rsMain.GetRobot j, R_bot.x, R_bot.y, R_bot.dir, R_bot.valid
              ' tmpstr = tmpstr + ":T:" + Format$(Ro_bot.X * (frmMain_FIELD_MAXX), "0") + ":" +
Format$(Ro_bot.Y * (frmMain_FIELD_MAXY), "0") + ":" + Format$(Ro_bot.dir * 180 / PI, "0")
              result = sendData(ANGEL, 0, CLng(R_bot.x * (frmMain_FIELD_MAXX)), CLng(R_bot.y *
(frmMain_FIELD_MAXY)), CLng(R_bot.dir * 180 / PI))
            End If
          Else
            rmMonitor.rsMain.SetRobot j, 0, 0, 0, False
          End If
        Next j

        frmMonitor.rsMain.Redraw
        UpdateCount = UpdateCount + 1
        ReceivedCount = 0
      End If
      End Select
End Sub
Private Sub tmrFind_Timer()
      ' Find client
      cmdFind_Click
      'findai_Click
End Sub
Private Sub tmrRequest_Timer()
```

```
    ' Request data from clients
    ReceivedCount = 0
    sockMain.RemoteHost = frmMain_BROADCAST
    sockMain.sendData "REQDAT"

    'Winsock1.RemoteHost = frmMain_BROADCAST
    'Winsock1.SendData "OK_START"
End Sub
Private Sub tmrReset_Timer()
    'Reset data
    Dim i As Long

    DataReceived(ReceivedCount).Ball.valid = False
    For i = 0 To 4
      DataReceived(ReceivedCount).Opponent(i).valid = False
      DataReceived(ReceivedCount).Robot(i).valid = False
    Next i
End Sub
Private Sub tmrRuntime_Timer()
    UpdateData ' Update form
End Sub
Private Function Distance(x1 As Double, y1 As Double, x2 As Double, y2 As Double) As Double    '
Return distance between 2 points
    Dim dx As Double, dy As Double
    dx = x2 - x1
    dy = y2 - y1
    Distance = Sqr((dx * dx) + (dy * dy))
End Function
Private Sub vsbFindClient_Change()
    txtFindClient.Text = Format$(vsbFindClient.Max + vsbFindClient.Min - vsbFindClient.Value, "0")
    frmMain_FINDCLIENT = Val(txtFindClient.Text) * 1000
```

```
    tmrFind.Interval = frmMain_FINDCLIENT
End Sub
Private Sub vsbRate_Change()
    txtRate.Text = Format$(vsbRate.Max + vsbRate.Min - vsbRate.Value, "0")
    frmMain_REQUEST = 1000 / Val(txtRate.Text)
    tmrRequest.Interval = frmMain_REQUEST
End Sub
```

ภาคผนวก ข

ซีซอร์สโค้ด

```
/*****************************************************
This program was produced by the
CodeWizardAVR V1.24.7e Evaluation
Automatic Program Generator
© Copyright 1998-2005 Pavel Haiduc, HP InfoTech s.r.l.
http://www.hpinfotech.com
e-mail:office@hpinfotech.com

Project :
Version :
Date    : 3/8/2007
Author  : Freeware, for evaluation and non-commercial use only
Company :
Comments:


Chip type           : ATmega128
Program type        : Application
Clock frequency     : 16.000650 MHz
Memory model        : Small
External SRAM size  : 0
Data Stack size     : 1024
*****************************************************/

#include <mega128.h>

// External Interrupt 0 service routine
```

```
interrupt [EXT_INT0] void ext_int0_isr(void)
{
// Place your code here

}

// External Interrupt 1 service routine
interrupt [EXT_INT1] void ext_int1_isr(void)
{
// Place your code here

}

// External Interrupt 2 service routine
interrupt [EXT_INT2] void ext_int2_isr(void)
{
// Place your code here

}

#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
```

```
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)

// Get a character from the USART1 Receiver
#pragma used+
char getchar1(void)
{
char status,data;
while (1)
    {
    while (((status=UCSR1A) & RX_COMPLETE)==0);
    data=UDR1;
    if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
      return data;
    };
}
#pragma used-

// Write a character to the USART1 Transmitter
#pragma used+
void putchar1(char c)
{
while ((UCSR1A & DATA_REGISTER_EMPTY)==0);
UDR1=c;
}
#pragma used-

// Standard Input/Output functions
#include <stdio.h>

// Declare your global variables here
```

```
unsigned char dat[8];
void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTA=0x00;
DDRA=0x00;

// Port B initialization
// Func7=Out Func6=Out Func5=Out Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=0 State6=0 State5=0 State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0xE0;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x00;

// Port E initialization
```

```
// Func7=In Func6=In Func5=Out Func4=Out Func3=Out Func2=In Func1=In Func0=In
// State7=T State6=T State5=0 State4=0 State3=0 State2=T State1=T State0=T
PORTE=0x00;
DDRE=0x38;

// Port F initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTF=0x00;
DDRF=0x00;

// Port G initialization
// Func4=In Func3=In Func2=In Func1=In Func0=In
// State4=T State3=T State2=T State1=T State0=T
PORTG=0x00;
DDRG=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
ASSR=0x00;
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 15.626 kHz
// Mode: Ph. correct PWM top=00FFh
```

```
// OC1A output: Non-Inv.
// OC1B output: Non-Inv.
// OC1C output: Non-Inv.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR1A=0xA9;
TCCR1B=0x05;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
OCR1CH=0x00;
OCR1CL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;
```

```
// Timer/Counter 3 initialization
// Clock source: System Clock
// Clock value: 15.626 kHz
// Mode: Ph. correct PWM top=00FFh
// Noise Canceler: Off
// Input Capture on Falling Edge
// OC3A output: Non-Inv.
// OC3B output: Non-Inv.
// OC3C output: Non-Inv.
// Timer 3 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR3A=0xA9;
TCCR3B=0x05;
TCNT3H=0x00;
TCNT3L=0x00;
ICR3H=0x00;
ICR3L=0x00;
OCR3AH=0x00;
OCR3AL=0x00;
OCR3BH=0x00;
OCR3BL=0x00;
OCR3CH=0x00;
OCR3CL=0x00;

// External Interrupt(s) initialization
// INT0: On
// INT0 Mode: Rising Edge
```

```
// INT1: On
// INT1 Mode: Rising Edge
// INT2: On
// INT2 Mode: Rising Edge
// INT3: Off
// INT4: Off
// INT5: Off
// INT6: Off
// INT7: Off
EICRA=0x3F;
EICRB=0x00;
EIMSK=0x07;
EIFR=0x07;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;
ETIMSK=0x00;

// USART0 initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART0 Receiver: On
// USART0 Transmitter: On
// USART0 Mode: Asynchronous
// USART0 Baud rate: 9600 at 0x67
UCSR0A=0x00;
UCSR0B=0x18;
UCSR0C=0x06;
UBRR0H=0x00;
UBRR0L=0x33;

// USART1 initialization
```

```
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART1 Receiver: On
// USART1 Transmitter: On
// USART1 Mode: Asynchronous
// USART1 Baud rate: 9600
UCSR1A=0x00;
UCSR1B=0x18;
UCSR1C=0x06;
UBRR1H=0x00;
UBRR1L=0x67;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// Global enable interrupts
#asm("sei")
 // code Robot 1 = A
 // code Robot 2 = B
 // code Robot 3 = C
 // code Robot 4 = D
 // code Robot 5 = E

while (1)
    {
    // Place your code here
    unsigned char m;
    unsigned char data_in;
     data_in=getchar();
```

```
if (data_in == 'A')
  {
  for(m=0;m<8;m++)
  {
      dat[m]=getchar();
  }
      OCR1AH=0x00;
      OCR1AL=dat[0];
      OCR1BH=0x00;
      OCR1BL=dat[1];
      OCR1CH=0x00;
      OCR1CL=dat[2];
      OCR3AH=0x00;
      OCR3AL=dat[3];
      OCR3BH=0x00;
      OCR3BL=dat[4];
      OCR3CH=0x00;
      OCR3CL=dat[5];
  }
 };
}
```

ภาคผนวก ค

Paper precent

บทความประกอบการสอบวิชา 01054025 Project 2 ภาคเรียนที่ 2/2550

หุ่นยนต์เตะฟุตบอล

(SOCCER ROBOT)

มณฑล ฉ่ำทรัพย์[1], กานต์ เงินเรืองนิรันดร์[1], ภูรี เทียมใส[1]

.....

ณัฐวุฒิ เดไปวา[2]

บทคัดย่อ

โครงงานนี้เป็นการศึกษาถึงการออกแบบและควบคุมหุ่นยนต์เตะฟุตบอลโดยใช้หลักการทฤษฎีทางด้านกลศาสตร์ พลศาสตร์รวมไปถึงหลักการทางคณิตศาสตร์ช่วยในการออกแบบและ ควบคุมหุ่นยนต์ด้วยไมโครคอนโทรลเลอร์ ซึ่งในการศึกษาถึงการทำงานของหุ่นยนต์เตะฟุตบอลก็เป็นอีกทางหนึ่งที่จะทำให้เข้าใจถึงการควบคุมการทำงานของหุ่นยนต์ โดยในโครงงานนี้การออกแบบโครงสร้างของหุ่นยนต์ ตัวหุ่นยนต์ ล้อและมอเตอร์ให้หุ่นยนต์มีอัตราเร่งที่ดี และอุปกรณ์อื่นๆ ของหุ่นยนต์ อย่างเช่น ระบบการเตะบอลจะเป็นการออกแบบโดยใช้ระบบโซลีนอยด์ และตัวเลี้ยงบอล ที่ใช้มอเตอร์ในการหมุนเลี้ยงบอล และจุดยึดของชิ้นส่วนต่างๆ การรับคำสั่งและประมวลผลโดยใช้ไมโครคอนโทรเลอร์ แปลผลสัญญาณจากคอมพิวเตอร์โดยหุ่นยนต์ในโครงงานนี้จะเป็นการพัฒนาจากหุ่นยนต์เตะฟุตบอลในปีการศึกษา 2549

Abstract

This project aim to comprehend robot mechanic through studying 'Robocup' by using mechanical discipline to design hardware. The project frame is to design the robot structure. Beside the movement system, the researchers also study about kicking system that use solenoid mechanism and for the dribbling mechanism we design the system to be able to calculate the best position for dribbling. For processing system, we use microcontroller system to process the signal from computer. This robot project is going to be soccer robot.

*Keywords:*Microcontroler;Solinoid,Soccerrobot,Dribbling

---

1. บทนำ

ในปัจจุบันเทคโนโลยีเข้ามามีผลต่อการดำเนินชีวิตประจำวันของมนุษย์ ดังจะเห็นโดยทั่วไปไม่ว่าจะเป็น เครื่องจักรในโรงงานอุตสาหกรรม เครื่องยนต์ เครื่องใช้ไฟฟ้า เครื่องมือช่าง เครื่องจักรกลและอื่น ๆ ในปัจจุบันอุปกรณ์เหล่านี้จะทำงานแบบกึ่งอัตโนมัติและทำงานแบบอัตโนมัติโดยจะถูกควบคุมการทำงานด้วยไมโครคอนโทรลเลอร์ ดังนั้นการศึกษาการควบคุมการทำงานของอุปกรณ์ต่าง ๆ ด้วยไมโครคอนโทรลเลอร์จึงจำเป็นในปัจจุบัน เพื่อนำไปแก้ไขปัญหาและพัฒนาอุปกรณ์เหล่านั้นให้มีประสิทธิภาพอีกต่อไป

ในปี พ.ศ. 2548 พิศาล และคณะ [1] ได้ออกแบบและสร้างหุ่นยนต์เตะฟุตบอลขึ้นเป็นครั้งแรก ต่อมาในปี พ.ศ. **2549** สามารถ และคณะ[2] ได้ออกแบบและสร้างหุ่นยนต์ขึ้นมาใหม่ โดยใช้การส่งสัญญาณแบบไร้สายและกล้องเพื่อประมวลผล จากผลการทดสอบการทำงานของหุ่นยนต์เตะฟุตบอลพบว่า หุ่นยนต์เคลื่อนที่ไม่สม่ำเสมอ เคลื่อนที่ออกนอกเส้นทางที่กำหนด ระบบยิงบอลเบา กล้องจับภาพช้า จากการที่ได้ศึกษาหุ่นยนต์เตะฟุตบอลแล้ว จึงได้นำปัญหาที่เกิดขึ้นมาแก้ไขและพัฒนาให้หุ่นยนต์มีความสามารถเพิ่มขึ้น

---

[1] นักศึกษาภาควิชาวิศวกรรมเครื่องกล สจล. ห้อง 3Q รหัส 48015388, 48015412และ 48015448ตามลำดับ

[2] อาจารย์ประจำภาควิชาวิศวกรรมเครื่องกล สจล., โทร. 086-7783931, อีเมล kdnattaw@kmitl.ac.th

2. **จุดประสงค์**

- เพื่อพัฒนาและสร้างหุ่นยนต์เตะฟุตบอลให้เคลื่อนที่เร็วขึ้น
- เพื่อพัฒนาหุ่นยนต์เตะฟุตบอลให้เคลื่อนที่ไปยังจุดที่เราต้องการได้
- เพื่อพัฒนาหุ่นยนต์เตะฟุตบอลให้มีฟังก์ชั่นการทำงานที่จะต้องใช้ในการเตะฟุตบอล
- เพื่อศึกษาการควบคุมการเคลื่อนที่ของหุ่นยนต์แบบมีล้อด้วยไมโครคอนโทรลเลอร์
- เพื่อศึกษาภารควบคุมแหล่งจ่ายไฟแบบ PWM โดยใช้ไมโครคอนโทรลเลอร์
- เพื่อศึกษาการควบคุมมอเตอร์โดยใช้ไมโครคอนโทรลเลอร์

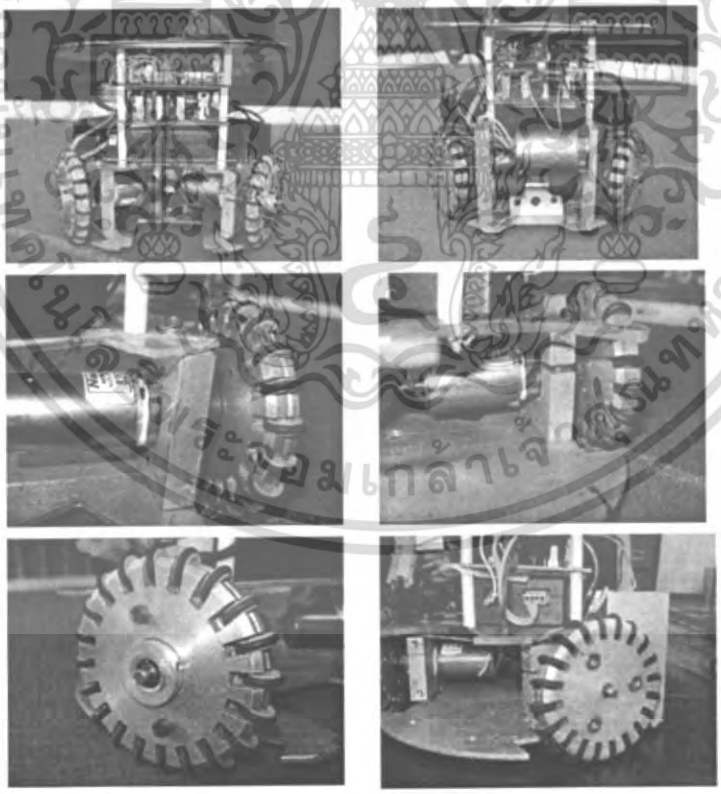
3. ขอบเขตของงานวิจัย

- หุ่นยนต์เตะฟุตบอลเคลื่อนที่ไปยังจุดที่เราต้องการได้ โดยทำงานแบบอัตโนมัติและเป็นทีม
- หุ่นยนต์เตะฟุตบอลมีขนาดเส้นผ่านศูนย์กลางไม่เกิน 180 มิลลิเมตร สูงไม่เกิน 150 มิลลิเมตร
- ใช้การส่งสัญญาณแบบไร้สาย
- การพัฒนาในส่วนของ AI จนสามารถทำให้หุ่นยนต์ทำงานกันเป็นทีม

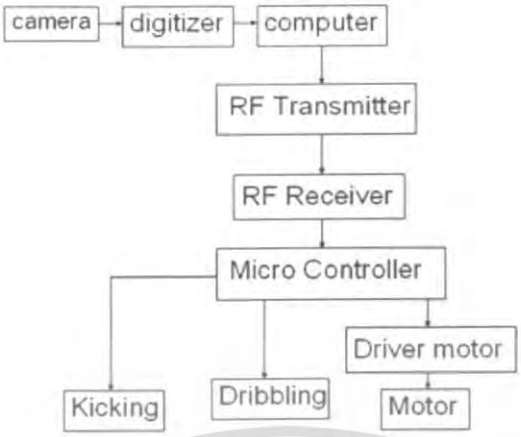
การออกแบบล้อ

4. ส่วนประกอบของหุ่นยนต์เตะฟุตบอล

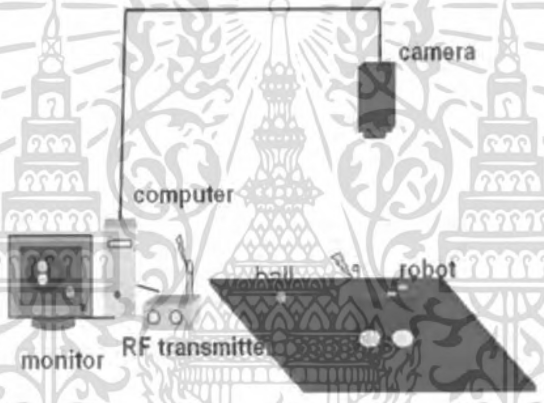
- ล้อ
- มอเตอร์กระแสตรงขับเคลื่อนล้อ
- มอเตอร์กระแสตรงเลี้ยงบอล
- เพลาและยางเลี้ยงบอล
- โซลีนอยด์ยิงบอล
- แบตเตอรี่
- โครงฐานของหุ่นยนต์



รูปด้านซ้ายคือหุ่นตัวปัจจุบัน



รูปที่ 5.1 หลักการทำงานของหุ่นยนต์



รูปที่ 5.2 แสดงโครงสร้างของระบบ

5. หลักการทำงาน

ในรูปที่ 5.1 แสดงหลักการทำงานของหุ่นยนต์เตะฟุตบอล เริ่มจากกล้องวิดีโอที่ติดอยู่เหนือสนามเก็บภาพของสนาม หุ่นยนต์และลูกบอล จากนั้นส่งข้อมูลภาพที่ได้ส่งให้คอมพิวเตอร์เพื่อที่คอมพิวเตอร์จะนำข้อมูลภาพที่ได้มาประมวลผลเพื่อควบคุมการเคลื่อนที่ของหุ่นยนต์ โดยจะส่งข้อมูลที่จะควบคุมหุ่นยนต์ผ่านพอร์ตอนุกรม และใช้ชุดรับส่งไร้สายในการส่งสัญญาณไปที่หุ่นยนต์ เพื่อให้ไมโครคอนโทรลเลอร์ AVR MEGA 128 นำข้อมูลที่ได้รับเปลี่ยนเป็นสัญญาณ **PWM** ไปควบคุมชุดขับมอเตอร์ (drive motor) เพื่อขับล้อ ชุดยิง (kicking) และชุดเลี้ยง (dribbling)

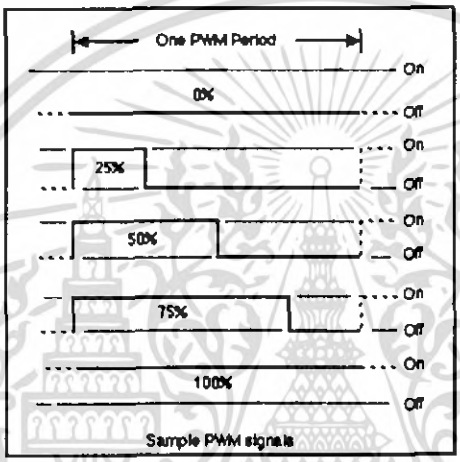
ขณะนี้ได้คิดค้นวิธีการควบคุมทิศทางการเคลื่อนที่ของหุ่นยนต์เป็นแบบโปรแกรม PWM มาใช้ควบคุมแหล่งจ่ายไฟโดยใช้ไมโครคอนโทรลเลอร์ และนำไปขับมอเตอร์ชุดขับเคลื่อน

การควบคุมแบบ PWM (Pulse Width Modulation)

คือ การควบคุมแรงดันให้กับโหลดเพื่อควบคุมความเร็ว โดยกำหนด Duty Cycle ของการ On และ Off เป็นเปอร์เซ็นต์

การทำงานของสัญญาณ PWM

- เส้นที่ 3 แสดงสัญญาณ PWM ที่ 50% duty cycle คือ สัญญาณในการออนจะเป็น 50% ของคาบสัญญาณ และ จะออฟเป็น 50% ของคาบสัญญาณ
- เส้นที่ 5 แสดงสัญญาณ PWM ที่ 100% duty cycle คือ สัญญาณในการออนจะเป็น 100% ของคาบสัญญาณ และ จะออฟเป็น 0% ของคาบสัญญาณ

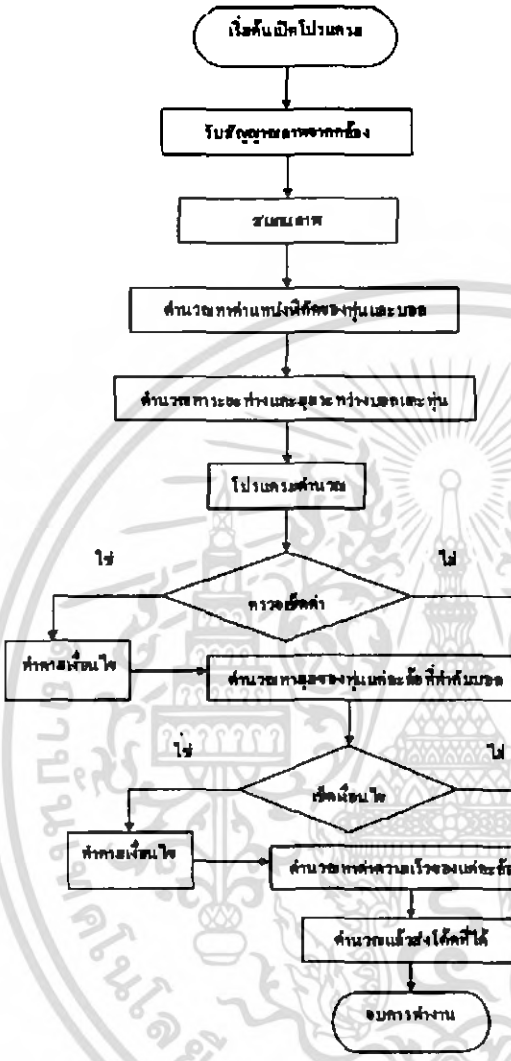


รูปแสดงสัญญาณ PWM ซึ่งแสดงค่า duty cycles ที่ต่างๆกัน

เหตุผลที่ใช้ PWM ในการควบคุมความเร็วมอเตอร์

- PWMง่ายในการอินเตอร์เฟสกับไมโครคอนโทรลเลอร์ และใช้เพียงแค่เอาท์พุตสัญญาณเดียวในการควบคุมความเร็ว
- ไมโครคอนโทรลเลอร์ที่ใช้มีฟังก์ชั่นในการทำงานแบบ PWM โดยไมโครคอนโทรลเลอร์จะเปลี่ยนสัญญาณอินท์พุตที่รับเข้ามาให้เป็นสัญญาณเอาท์พุตแบบ PWM
- เมื่อมีการควบคุมกำลังของแต่ละล้อจะทำให้หุ่นยนต์เคลื่อนที่ ในทิศทางต่างๆได้อย่างราบเรียบและแม่นยำ

หลักการทำงานในส่วนของโปรแกรมวิชชวลเบสิก



การคำนวณหาทิศทางและตำแหน่งของหุ่นยนต์แบบใหม่

del_y = (62 -31) = 31

del_x = (25 - 55) = -30

กรณี

If (del_y1 < 0 And del_x1 < 0) Or (del_y1 > 0 And del_x1 < 0)

$\beta eta = \beta eta + 180$

$\beta eta = \tan^{-}\left(\frac{31}{-30}\right) = -45$

$= -45 + 180 = 135$

$Zeta = \beta eta - \deg rees robot \quad = 135 - 90 = 45$

กรณี

$Zeta < -180$

$Zeta = 360 + Zeta$

If $Zeta > 180$

$Zeta = Zeta - 360$

จาก มุมล้อที่1 = Zeta - 30

มุมล้อที่ 2 = Zeta - 150

มุมล้อที่ 3 = Zeta - 270

มุมล้อที่1 = 45 - 30 = 15

มุมล้อที่ 2 = 45 - 150 = -105

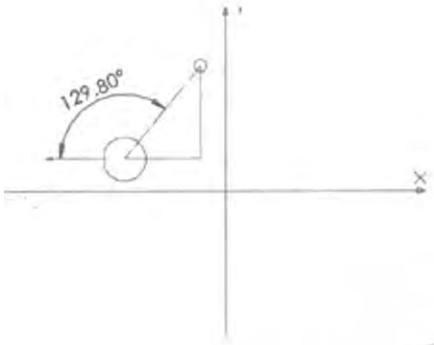
มุมล้อที่ 3 = 45 - 270 = -225

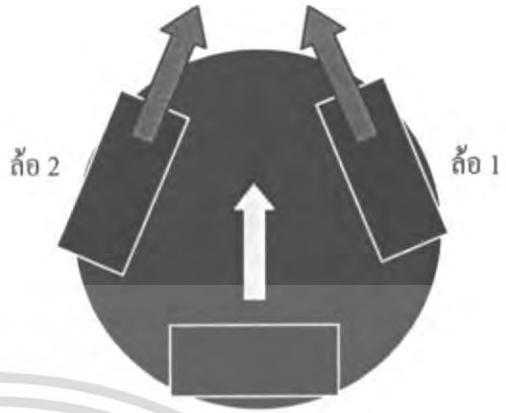
ความเร็วล้อที่1 = cos มุมล้อที่1 = cos (15) = 0.96 × 100 % = 96%

ความเร็วล้อที่2 = cos มุมล้อที่2 = cos (-105) = 0.26 × 100 % = 26%

ความเร็วล้อที่3 = cos มุมล้อที่3 = cos (-225) = -0.71 × 100 % = -71%

การคำนวณหาทิศทางและตำแหน่งของหุ่นยนต์แบบเดิม





ระยะขจัดแนวแกน X = X2 – X1

= -5 - (-10) = 5

ระยะขจัดแนวแกน Y = Y2 – Y1

= 12 - 6 = 6

$$\beta eta = \tan^{-1}\left(\frac{Y}{X}\right)$$

$$= \tan^{-1}\left(\frac{6}{5}\right) = 50.2 \text{ องศา}$$

เข้ากรณี X > 0 และ Y > 0

กำหนด Alpha = βeta

จะได้ Alpha = 50.2 องศา

กำหนด Zeta = Alpha - Grammar

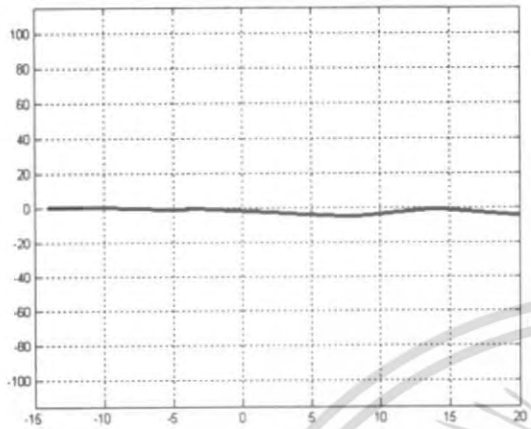
จะได้ Zeta = 50.2 – 180

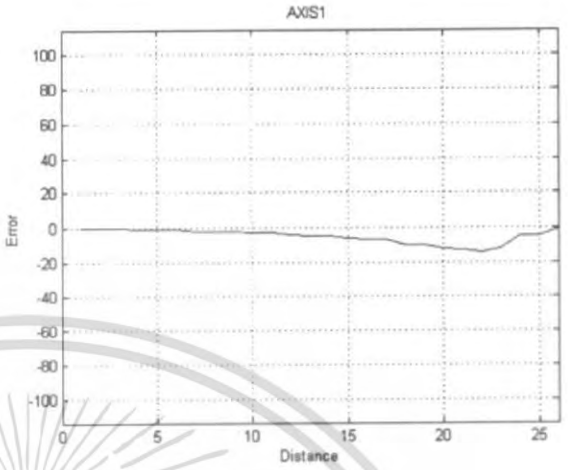
Zeta = -129.8 องศา

ได้มุม Zeta มีค่าเป็น ลบ แสดงว่าหุ่นหมุนตามเข็ม 129.8 องศา

เมื่อหุ่นหมุนหันหน้าเข้าหาบอลแล้ว ล้อที่ 1 กับล้อที่ 2 จะมีความเร็วเท่ากันแต่มีทิศทางการหมุนตรงข้ามกัน ส่วนล้อที่ 3 จะไม่หมุน

กราฟแสดงผลการทดลองการเดินทางไปยังเป้าหมายของหุ่น





การทดลองครั้งใหม่ การทดลองครั้งเก่า

จากการทดลองพบว่าเมื่อหุ่นเคลื่อนที่ไปได้ระยะหนึ่งหน้าหุ่นจะมีการเบี่ยงเบนออกจากเป้าหมายแต่หุ่นก็จะสามารถควบคุมตัวเองให้หันหน้าและเคลื่อนที่เข้าหาเป้าหมายได้อย่างดีขึ้น

รูปหุ่นยนต์ FREEDOM+ TEAM

สรุปผลการทดลอง

- หุ่นยนต์มีขนาดตามกติกาที่กำหนด
- หุ่นยนต์สามารถวิ่งเข้าหาบอลได้อย่างอัตโนมัติและเป็นทีม
- ในบางครั้งหุ่นอาจจะมีการเคลื่อนที่เบี่ยงเบนออกจากบอลไปบ้างแต่ก็สามารถวิ่งหันหน้ากลับเข้าหาบอลได้ เพราะการทำงานของระบบเป็นระบบปิด
- บางครั้งหุ่นมีการหยุดชะงักขึ้นทำให้การเคลื่อนที่โดยรวมไม่ค่อยราบเรียบเป็นเพราะกล้องจับภาพไม่ได้ในขั่วขณะนั้น
- ในกรณีที่ปล่อยหุ่นลงสนามพร้อมกันหลายตัวยิ่งทำให้สัญญาณในการส่งจะช้าลง ทำให้ประมวลผลก็ช้าลงด้วย

แนวทางแก้ไขและพัฒนา

- เนื่องจากกล้องที่ใช้มีคุณภาพต่ำทำให้การอ่านสีจะมีความผิดพลาดอยู่เสมอ
- กล้องที่ใช้ก็มีเพียงตัวเดียวทำให้ไม่สามารถมองเห็นได้ทั่วสนาม ดังนั้นจึงควรใช้กล้องไม่ต่ำกว่า 2 ตัว เพื่อที่จะมองเห็นได้ทั่วสนาม
- จำนวนคอมพิวเตอร์ที่ใช้อย่างน้อยต้องไม่ต่ำกว่าจำนวนกล้องที่ติดเพื่อที่การประมวลผลจะมีความรวดเร็วและแม่นยำยิ่งขึ้น
- ควรใช้ตัวรับส่งสัญญาณแบบตัวต่อตัวเพื่อป้องกันการสับสนของหุ่นและจะช่วยให้ประสิทธิภาพในการส่งสัญญาณดีขึ้น
- ในส่วนของปัญญาประดิษฐ์ (AI) จะมีความซับซ้อนมากจึงควรมีการพัฒนาต่อไป

เอกสารอ้างอิง

[1] พิศาล มูลอำคา, พุฒิพงศ์ มีประทัง, จิรายุทธ คงโท, หุ่นยนต์เตะฟุตบอล, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2548

[2] วิทยา วงษ์กลาง, สามารถ มุ่งโตกลาง, ไชสาร ทรงประโคนหุ่นยนต์เตะฟุตบอล, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2549

[3] มงคล เทียนวิบูลย์, การควบคุมหุ่นยนต์เคลื่อนที่ด้วยล้อที่สามารถเคลื่อนที่ได้อย่างอิสระ, วิทยานิพนธ์ปริญญามหาบัณฑภาควิชาวิศวกรรมเครื่องกล คณะวิศวกรรมศาสตร์, จุฬาลงกรณ์มหาวิทยาลัย, 2542

[4] Patrick F. Muir and Charles P. Neuman, Kinematic Modeling of Wheeled Mobile Robots, Ph.D Dissertation. Department of Electrical and Computer Engineering, The Robotics Institute, Carnegie Mellon University, 1988.

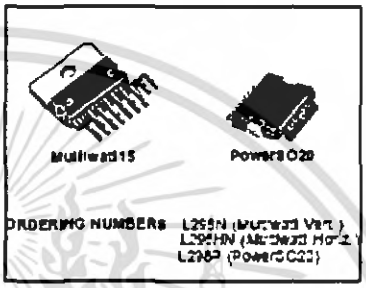
ภาคผนวก ง

Data Sheet L298N

L298

DUAL FULL-BRIDGE DRIVER

- OPERATING SUPPLY VOLTAGE UP TO 46 V
- TOTAL DC CURRENT UP TO 4 A
- LOW SATURATION VOLTAGE
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)

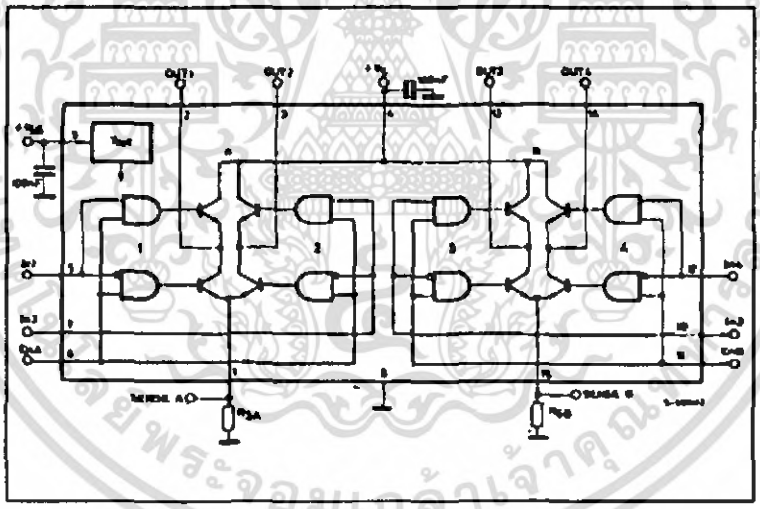


ORDERING NUMBERS: L298N (Multiwatt Vert.)
L298HN (Multiwatt Horiz.)
L298P (PowerSO20)

DESCRIPTION

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

BLOCK DIAGRAM



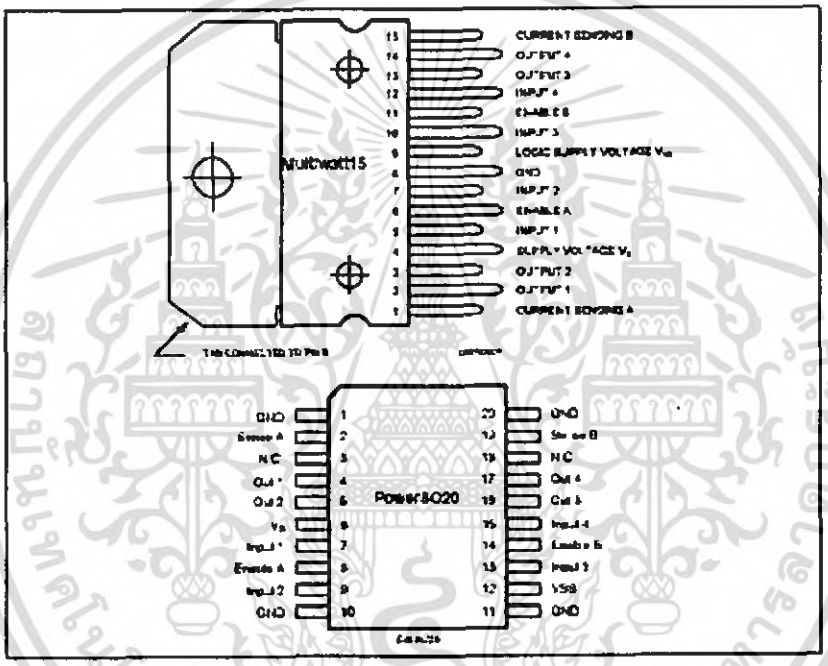
January 2000

1/13

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_i, V_{en}	Input and Enable Voltage	−0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive (t = 100µs) –Repetitive (50% on –20% off, t_p = 10ms) –DC Operation	3 2.5 2	A A A
V_{sens}	Sensing Voltage	−1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75°C)	25	W
T_{op}	Junction Operating Temperature	−25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	−40 to 150	°C

PIN CONNECTIONS (top view)



THERMAL DATA

Symbol	Parameter		PowerSO20	Multiwatt15	Unit
$R_{th\,j\text{-}case}$	Thermal Resistance Junction-case	Max.	–	3	°C/W
$R_{th\,j\text{-}amb}$	Thermal Resistance Junction-ambient	Max.	13 (*)	35	°C/W

(*) Mounted on aluminium substrate

PIN FUNCTIONS (refer to the block diagram)

MW.15	PowerSO	Name	Function
1, 15	2, 19	Sense A, Sense B	Between this pin and ground is connected the sense resistor to control the current of the load.
2, 3	4, 5	Out 1, Out 2	Outputs of the Bridge A. The current that flows through the load connected between these two pins is monitored at pin 1.
4	6	V_S	Supply Voltage for the Power Output Stages. A non-inductive 100nF capacitor must be connected between this pin and ground.
5, 7	7, 9	Input 1, Input 2	TTL Compatible Inputs of the Bridge A.
6, 11	8, 14	Enable A, Enable B	TTL Compatible Enable Input: the L state disables the bridge A (enable A) and/or the bridge B (enable B).
8	1, 10, 11, 20	GND	Ground.
9	12	V_{SS}	Supply Voltage for the Logic Blocks. A 100nF capacitor must be connected between this pin and ground.
10, 12	13, 15	Input 3, Input 4	TTL Compatible Inputs of the Bridge B.
13, 14	16, 17	Out 3, Out 4	Outputs of the Bridge B. The current that flows through the load connected between these two pins is monitored at pin 15.
–	3, 18	N.C.	Not Connected.

ELECTRICAL CHARACTERISTICS (V_S = 42V, V_{SS} = 5V, T_j = 25°C; unless otherwise specified)

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
V_S	Supply Voltage (pin 4)	Operative Condition		V_{IH} +2.5		46	V
V_{SS}	Logic Supply Voltage (pin 9)			4.5	5	7	V
I_S	Quiescent Supply Current (pin 4)	V_{en} = H; I_L = 0	V_i = L V_i = H		13 50	22 70	mA mA
		V_{en} = L	V_i = X			4	mA
I_{SS}	Quiescent Current from V_{SS} (pin 9)	V_{en} = H; I_L = 0	V_i = L V_i = H		24 7	36 12	mA mA
		V_{en} = L	V_i = X			6	mA
V_{il}	Input Low Voltage (pins 5, 7, 10, 12)			-0.3		1.5	V
V_{ih}	Input High Voltage (pins 5, 7, 10, 12)			2.3		V_{SS}	V
I_{il}	Low Voltage Input Current (pins 5, 7, 10, 12)	V_i = L				-10	µA
I_{ih}	High Voltage Input Current (pins 5, 7, 10, 12)	V_i = H ≤ V_{SS} -0.6V			30	100	µA
V_{en} = L	Enable Low Voltage (pins 6, 11)			-0.3		1.5	V
V_{en} = H	Enable High Voltage (pins 6, 11)			2.3		V_{SS}	V
I_{en} = L	Low Voltage Enable Current (pins 6, 11)	V_{en} = L				-10	µA
I_{en} = H	High Voltage Enable Current (pins 6, 11)	V_{en} = H ≤ V_{SS} -0.6V			30	100	µA
$V_{CE(sat)H}$	Source Saturation Voltage	I_L = 1A I_L = 2A		0.95	1.35 2	1.7 2.7	V V
$V_{CE(sat)L}$	Sink Saturation Voltage	I_L = 1A (5) I_L = 2A (5)		0.85	1.2 1.7	1.6 2.3	V V
$V_{CE(sat)}$	Total Drop	I_L = 1A (5) I_L = 2A (5)		1.80		3.2 4.8	V V
V_{sens}	Sensing Voltage (pins 1, 15)			-1 (1)		2	V

ELECTRICAL CHARACTERISTICS (continued)

Symbol	Parameter	Test Conditions	Min	Typ	Max	Unit
T_1 (V_i)	Source Current Turn-off Delay	0.5 V_i to 0.9 I_L (2), (4)		1.5		µs
T_2 (V_i)	Source Current Fall Time	0.9 I_L to 0.1 I_L (2), (4)		0.2		µs
T_3 (V_i)	Source Current Turn-on Delay	0.5 V_i to 0.1 I_L (2), (4)		2		µs
T_4 (V_i)	Source Current Rise Time	0.1 I_L to 0.9 I_L (2), (4)		0.7		µs
T_5 (V_i)	Sink Current Turn-off Delay	0.5 V_i to 0.9 I_L (3), (4)		0.7		µs
T_6 (V_i)	Sink Current Fall Time	0.9 I_L to 0.1 I_L (3), (4)		0.25		µs
T_7 (V_i)	Sink Current Turn-on Delay	0.5 V_i to 0.9 I_L (3), (4)		1.6		µs
T_8 (V_i)	Sink Current Rise Time	0.1 I_L to 0.9 I_L (3), (4)		0.2		µs
fc (V_i)	Commutation Frequency	I_L = 2A		25	40	kHz
T_1 (V_{en})	Source Current Turn-off Delay	0.5 V_{en} to 0.9 I_L (2), (4)		3		µs
T_2 (V_{en})	Source Current Fall Time	0.9 I_L to 0.1 I_L (2), (4)		1		µs
T_3 (V_{en})	Source Current Turn-on Delay	0.5 V_{en} to 0.1 I_L (2), (4)		0.3		µs
T_4 (V_{en})	Source Current Rise Time	0.1 I_L to 0.9 I_L (2), (4)		0.4		µs
T_5 (V_{en})	Sink Current Turn-off Delay	0.5 V_{en} to 0.9 I_L (3), (4)		0.2		µs
T_6 (V_{en})	Sink Current Fall Time	0.9 I_L to 0.1 I_L (3), (4)		0.35		µs
T_7 (V_{en})	Sink Current Turn-on Delay	0.5 V_{en} to 0.9 I_L (3), (4)		0.25		µs
T_8 (V_{en})	Sink Current Rise Time	0.1 I_L to 0.9 I_L (3), (4)		0.1		µs

*) If a sensing voltage is not used, sense terminal must be directly connected to ground.
2) See fig. 2.
3) See fig. 4.
4) The load must be a pure resistor.

Figure 1 : Typical Saturation Voltage vs Output Current.

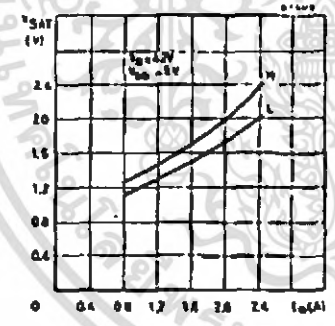
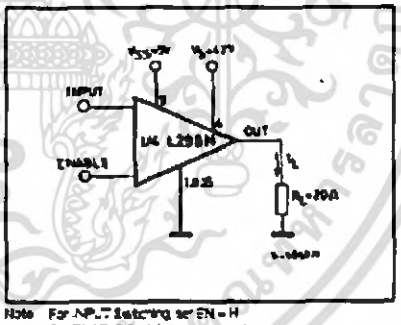


Figure 2 : Switching Times Test Circuits.



Note: For INPUT Switching set EN = H
For ENABLE Switching set IN = H

Figure 3 : Source Current Delay Times vs. Input or Enable Switching

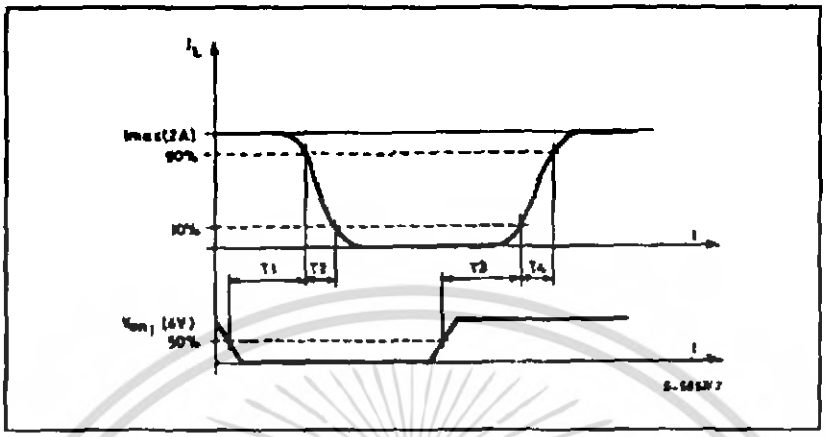
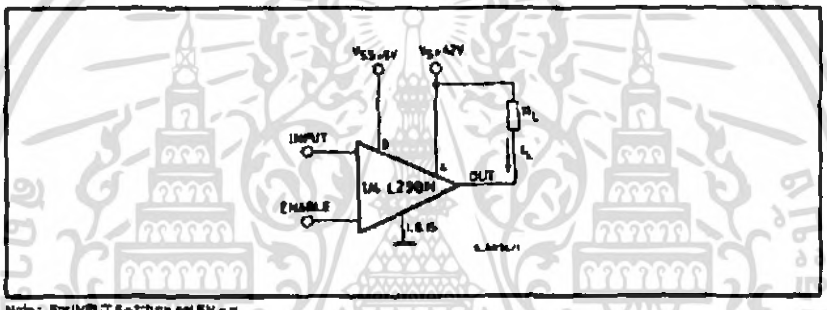


Figure 4 : Switching Times Test Circuits



Note : For INPUT Switching, set EN = H
For ENABLE Switching, set IN = L