รถวิ่งตามเส้นโดยใช้การประมวลผลสัญญาณภาพ

LINE-TRACKING ROVER USING IMAGE PROCESSING

โดย

นายอธิวัชร ถนอมพุทรา

นายอธิวัฒน์ เฉียบแหลม

นายอภิวงศ์ พรหมกัลป์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2556

รถวิ่งตามเส้นโดยใช้การประมวลผลสัญญาณภาพ
LINE-TRACKING ROVER USING IMAGE PROCESSING

โดย

นายอธิวัชร ถนอมพุทรา

นายอธิวัฒน์ เฉียบแหลม

นายอภิวงศ์ พรหมกัลป์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2556

รถวิ่งตามเส้นโดยใช้การประมวลผลสัญญาณภาพ

LINE-TRACKING ROVER USING IMAGE PROCESSING

โดย

นายอธิวัชร	ถนอมพุทรา	53011832
นายอธิวัฒน์	เฉียบแหลม	53011833
นายอภิวงศ์	พรหมกัลป์	53011868

อาจารย์ที่ปรึกษา

รศ.ดร.ไกรสิน ส่งวัฒนา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2556

ผ่านการตรวจรูปเล่มแล้ว

(.....)

อาจารย์ที่ปรึกษา

11/3/57

KMITL วิศวกรรมโทรคมนาคม Telecommunications Engineering

ผ่านการตรวจชิ้นงานแล้ว

(.....)

กรรมการผู้ตรวจชิ้นงาน

14/03/14

KMITL วิศวกรรมโทรคมนาคม

ปริญญานิพนธ์ปีการศึกษา 2556
สาขาวิชาวิศวกรรมโทรคมนาคม
คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
เรื่อง รถวิ่งตามเส้นโดยใช้การประมวลผลสัญญาณภาพ
LINE-TRACKING ROVER USING IMAGE PROCESSING

ผู้จัดทำ

1 นายอธิวัชร	ถนอมพุทรา	53011832
2 นายอธิวัฒน์	เฉียบแหลม	53011833
3 นายอภิวงศ์	พรหมกัลป์	53011868

..... อาจารย์ที่ปรึกษา
(รศ.ดร.ไกรสิน ส่งวัฒนา)

กิตติกรรมประกาศ

โครงงานได้สำเร็จลุล่วงเป็นอย่างดี ต้องขอขอบพระคุณ รศ.ดร.ไกรสิน ส่งวัฒนา ซึ่งเป็นอาจารย์ที่ปรึกษาโครงงาน ที่ให้คำแนะนำชี้แนวทางในการแก้ปัญหาและให้คำปรึกษาตลอดระยะเวลาที่ทำโครงงานและเขียนรูปเล่มรายงาน นอกจากนี้ยังให้ความอนุเคราะห์ในการใช้สถานที่สำหรับทำโครงงาน ผู้จัดทำโครงงานรู้สึกซาบซึ้งในความอนุเคราะห์เมตตาจากท่านอาจารย์และกราบขอบพระคุณเป็นอย่างสูง นอกจากนี้ขอขอบคุณบิดามารดา รวมทั้งขอบคุณพี่ๆ และเพื่อนๆ นักศึกษา ที่คอยให้กำลังใจและให้คำแนะนำ ชี้แนะแนวทาง ตลอดระยะเวลาการทำโครงงานครั้งนี้ จนโครงงานฉบับนี้สำเร็จลุล่วงเป็นอย่างดี

สุดท้ายนี้ทางคณะผู้จัดทำหวังเป็นอย่างยิ่งว่า โครงงานฉบับนี้จะเป็นประโยชน์แก่ผู้ที่สนใจได้บ้างตามสมควร หากมีข้อเสนอแนะประการใดเพื่อปรับปรุงผลงานให้ดีขึ้น ทางคณะผู้จัดทำขอน้อมรับคำเสนอแนะด้วยความขอบพระคุณยิ่ง

นายอธิวัชร ถนอมพุทรา
นายอธิวัฒน์ เฉียบแหลม
นายอภิวงศ์ พรหมกัลป์

ผู้จัดทำ

รถวิ่งตามเส้นโดยใช้การประมวลผลสัญญาณภาพ

LINE-TRACKING ROVER USING IMAGE PROCESSING

โดย 1. อธิวัชร ถนอมพุทรา 53011832
2. อธิวัฒน์ เฉียบแหลม 53011833
3. อภิวงศ์ พรหมกัลป์ 53011868

อาจารย์ที่ปรึกษา รศ.ดร.ไกรสิน ส่งวัฒนา

บทคัดย่อ

โครงงานนี้เกี่ยวกับการพัฒนาการควบคุมรถในระบบอัตโนมัติโดยใช้การใช้ภาพจากกล้องแทนตาในการมองเห็นของมนุษย์ โดยจะใช้ภาพที่ได้ส่งไปที่คอมพิวเตอร์ส่วนประมวลผลเพื่อใช้ในการบังคับทิศทางการเคลื่อนที่ของรถ และส่งไปยังส่วนของไมโครคอนโทรลเลอร์ในการควบคุมวงจรมอเตอร์กระแสตรงเพื่อให้รถเคลื่อนที่ได้อย่างเหมาะสม

ABSTRACT

The aim of this project is to build automatic vehicle that can be controlled by image processing technique. A real time image, via a camera, is processed by computer for locating road line for vehicle. The position of the line is transmitted to a programmed microcontroller for controlled DC motor circuit for controlling the position of vehicle.

สารบัญ

สารบัญ (ต่อ)

สารบัญรูป

หน้า

รูปที่

สารบัญรูป (ต่อ)

หน้า

รูปที่

สารบัญตาราง

หน้า

ตารางที่

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

เทคโนโลยีในปัจจุบันสามารถทำให้การใช้ชีวิตประจำวันของมนุษย์สะดวกสบายยิ่งขึ้น โดยเฉพาะเทคโนโลยีที่เป็นระบบอัตโนมัติที่ใช้ทรัพยากรมนุษย์ลดลงและใช้เครื่องจักรทำงานแทนมากขึ้น และเทคโนโลยีการบังคับรถอัตโนมัติ เป็นอีกเทคโนโลยีที่น่าสนใจ ที่สามารถใช้ได้ในด้านความช่วยเหลือกิจกรรมต่างๆ เช่น การดูแลรักษาความปลอดภัย การขนส่งของ และเหตุการณ์ต่างๆ ที่เกิดขึ้น

โครงงานนี้เกี่ยวกับการพัฒนาการควบคุมรถในระบบอัตโนมัติโดยใช้ภาพจากกล้องแทนตาในการมองเห็นของมนุษย์ โดยจะใช้ภาพที่ได้ส่งไปที่คอมพิวเตอร์ส่วนประมวลผลเพื่อใช้ในการบังคับทิศทางการเคลื่อนที่ของรถ และส่งไปยังส่วนของไมโครคอนโทรลเลอร์ในการควบคุมวงจรมอเตอร์กระแสตรงเพื่อให้รถเคลื่อนที่ได้อย่างเหมาะสม

1.2 วัตถุประสงค์

1. เพื่อใช้งานไมโครคอนโทรลเลอร์
2. เพื่อศึกษา Image Processing
3. เพื่อสร้างรถที่สามารถวิ่งตามเส้นทางที่กำหนด เช่น ภายในโรงงาน เป็นต้น
4. เพื่อประยุกต์ใช้งานการสื่อสารแบบไร้สาย
5. เพื่อแก้ไขความยุ่งยากของการใช้สายนำสัญญาณ

1.3 ขอบเขตของโครงงาน

สามารถพัฒนารถวิ่งตามเส้นโดยใช้การประมวลสัญญาณภาพ โดยใช้ภาพจากกล้องส่งเข้าไปคอมพิวเตอร์ประมวลเป็นคำสั่งส่งไปที่ไมโครคอนโทรลเลอร์เพื่อควบคุมวงจรขับมอเตอร์กระแสตรงที่จะทำให้รถสามารถเคลื่อนที่ได้

บทที่ 2

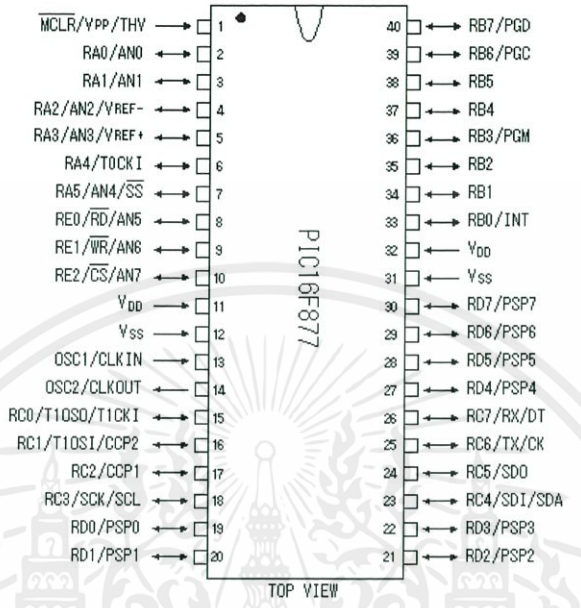
ทฤษฎีและหลักการที่เกี่ยวข้อง

2.1 ไมโครคอนโทรลเลอร์ PIC26F877

2.1.1 คุณลักษณะพื้นฐานของ PIC16F877

1. มีคำสั่งให้ใช้งานได้ 35 คำสั่ง
2. คำสั่งหนึ่งๆใช้เวลาทำงาน 1 ถึง cycle
3. ทำงานได้สูงสุดที่สัญญาณนาฬิกาตั้งแต่ไฟตรงถึง20MHz
4. ทำงานแบบ pipe-lineทำให้สามารถทำงานพร้อมกันในเวลาเดียวกันได้
5. หน่วยความจำโปรแกรมเป็นแบบ Flash มีขนาด8K Word
6. มีหน่วยความจำข้อมูล (Data Memory RAM) ขนาด 368 ไบต์
7. มีหน่วยความจำแบบ EEPROM ขนาด 256 ไบต์
8. ตอบสนองกับอินเตอร์รัพต์ได้ทั้งหมด 14 แหล่ง
9. มี Stack ให้ใช้ได้สูงสุด 8 ระดับ
10. มีระบบ Power On Reset, Power Up Timer, Oscillator Start-up timer
11. มีระบบ Watchdog timer
12. มีระบบ Code Protection ป้องกันการคัดลอกเลียนแบบ
13. มีโหมดประหยัดพลังงาน (Sleep mode)
14. สัญญาณนาฬิกามีหลายโหมดให้เลือกใช้งาน เช่น XTAL และ วงจรRC
15. สามารถโปรแกรมด้วยไฟ +5VDC ได้
16. ใช้การโปรแกรมแบบ In-Circuit Serial Programming
17. ทำงานที่ไฟเลี้ยง 2VDC ถึง 5.5VDC
18. Current Sink และ Current Source อยู่ที่ 25mA
19. มีTimer/Counter 3 ตัว
20. มีโมดูล Capture/Compare/PWM อีก2ชุด
21. มีA/D Converter แบบ 10 บิต
22. มีระบบ USART สำหรับต่อกับการสื่อสารแบบ RS232
23. มีระบบตรวจระดับไฟเลี้ยง (Brown-out reset)
24. มี I/O พอร์ตทั้งหมด 5 พอร์ต

2.1.2 โครงสร้างขาสัญญาณของไมโครคอนโทรลเลอร์ PIC16F877



รูปที่ 2.1 การจัดขามาตรฐานของไมโครคอนโทรลเลอร์ PIC16F877

ไมโครคอนโทรลเลอร์ตระกูลPICเบอร์PIC16F877 เป็นไมโครคอนโทรลเลอร์ขนาด 40 ขา มีขาสัญญาณต่างๆดังนี้

1. MCLR/Vpp : Master Clear(Reset) Input/Programming Voltage Input ทำหน้าที่เป็นสัญญาณรีเซต(Reset)เมื่อขานี้ได้รับลอจิก 0 ไมโครคอนโทรลเลอร์จะถูกรีเซต และทำหน้าที่เป็นขาสัญญาณรับแรงดัน ขณะทำการบันทึกโปรแกรมลงหน่วยความจำของไมโครคอนโทรลเลอร์

2. VDD : Positive Supply (+2.00V ถึง +5.5V)ทำหน้าที่เป็นขาไฟเลี้ยง

3. VSS : Ground ทำหน้าที่เป็นขากราวน์

4. OSC1/CLKIN : Oscillator Crystal Input/External clock Source Input

5. OSC2/CLKOUT : Oscillator Crystal Input/External clock Source Output ทั้งสองขาทำหน้าที่เป็นขาสัญญาณสำหรับคริสตัล ในกรณีที่อยู่ในโหมดการใช้สัญญาณนาฬิกาจากภายนอก(Crystal Oscillator Mode)

6. RA0 - RA5 : พอร์ต A มีจำนวน 6 ขา เป็นพอร์ตแบบสองทิศทาง(Bi-directional I/O Port)คือ เป็นได้ทั้งพอร์ตอินพุตและพอร์ตเอาต์พุตใช้ในการรับและการส่งข้อมูล นอกจากนี้ยังทำหน้าอื่นๆแสดงดังตารางที่ 2.1

ตารางที่ 2.1 หน้าที่ของขาสัญญาณของพอร์ต A

พอร์ต	สัญญาณ	หน้าที่
RA0	AN0	รับสัญญาณอินพุตสำหรับสำหรับ ADC ช่อง 0
RA1	AN1	รับสัญญาณอินพุตสำหรับสำหรับ ADC ช่อง 1
RA2	AN2	รับสัญญาณอินพุตสำหรับสำหรับ ADC ช่อง 2
RA3	AN3	รับสัญญาณอินพุตสำหรับสำหรับ ADC ช่อง 3
RA4	TOCK1	รับสัญญาณ Input Clock ของ Timer 0
RA5	AN4	รับสัญญาณอินพุตสำหรับสำหรับ ADC ช่อง 4
	SS	รับสัญญาณ Slave Select 0 จากการติดต่อของ Serial Port แบบ Synchronize

7. RB0 - RB7 : พอร์ตB มีจำนวน 8 ขา ขนาด 8 บิต เป็นพอร์ตแบบสองทิศทาง ใช้ในการส่งและรับข้อมูล นอกจากนี้บางขายังทำหน้าที่รับสัญญาณอินพุตจากการอินเตอร์รัปต์ (Interrupt) จากภายนอกด้วย แสดงดังตารางที่ 2.2

ตารางที่ 2.2 หน้าที่ของขาสัญญาณของพอร์ต B

พอร์ต	สัญญาณ	หน้าที่
RB0	INT	รับสัญญาณอินพุตจากการอินเตอร์รัพต์จากภายนอก
RB3	PGM	รับสัญญาณอินพุตแรงดันต่ำในการบันทึกโปรแกรม(ถ้ามีการ Enable)
RB6	PGC	ขาสัญญาณนาฬิกาในการบันทึกโปรแกรม
RB7	PGD	ขาสัญญาณข้อมูลในการบันทึกโปรแกรม

8. RC0 - RC7 : พอร์ต C มีจำนวน 8 ขา ขนาด 8 บิต เป็นพอร์ตแบบสองทิศทางใช้ในการส่งและรับข้อมูล นอกจากนี้ยังทำหน้าที่อื่นๆ แสดงดังตารางที่ 2.3

ตารางที่ 2.3 หน้าที่ของขาสัญญาณของพอร์ต C

พอร์ต	สัญญาณ	หน้าที่
RC0	T1OSO	ขาสัญญาณเอาต์พุตของวงจรออสซิลเลเตอร์ของ Timer
	TICK1	ขาสัญญาณอินพุตของสัญญาณนาฬิกาของ Timer 1
RC1	T1OSI	ขาสัญญาณอินพุตของวงจรออสซิลเลเตอร์ของ Timer 1
	CCP2	ขาสัญญาณเอาต์พุตของโมดุล CCP 2 (Cature2,compare2,PWM2)
RC2	CCP1	ขาสัญญาณเอาต์พุตของโมดูล CCP 1 (Cature1,Compare1,PWM1)
RC3	SCK	ขาสัญญาณนาฬิกาของวงจร SPI
	SCL	ขาสัญญาณนาฬิกาของวงจร I2C
RC4	SDI	ขาสัญญาณอินพุตและ Serial Data ของระบบ SPI
	SDA	ขาข้อมูลของระบบบัส I2C
RC5	SDO	ขาสัญญาณเอาต์พุตและ Serial Data ของระบบ SPI
RC6	TxD	ขาส่งข้อมูลแบบ Serial Port
	CK	ขาสัญญาณนาฬิกาแบบ Synchronize
RC7	RxD	ขารับข้อมูลแบบ Serial Port
	DT	ขาข้อมูลแบบ Synchronize

9. RD0 – RD7 : พอร์ต A มีจำนวน 8 ขา ขนาด 8 บิต เป็นพอร์ตแบบสองทิศทาง ใช้ในการรับส่งข้อมูล นอกจากนี้ยังมีหน้าที่อื่นๆ แสดงดังตารงที่ 2.4

ตารางที่ 2.4 หน้าที่ของขาสัญญาณของพอร์ต D

พอร์ต	สัญญาณ	หน้าที่
RD0	PSP0	ขาสัญญาณขยายพอร์ตแบบขนาน บิต 0
RD1	PSP1	ขาสัญญาณขยายพอร์ตแบบขนาน บิต 1
RD2	PSP2	ขาสัญญาณขยายพอร์ตแบบขนาน บิต 2
RD3	PSP3	ขาสัญญาณขยายพอร์ตแบบขนาน บิต 3
RD4	PSP4	ขาสัญญาณขยายพอร์ตแบบขนาน บิต 4
RD5	PSP5	ขาสัญญาณขยายพอร์ตแบบขนาน บิต 5
RD6	PSP6	ขาสัญญาณขยายพอร์ตแบบขนาน บิต 6
RD7	PSP7	ขาสัญญาณขยายพอร์ตแบบขนาน บิต 7

10. RE0-RE2 : พอร์ต E มีขาจำนวน 3 ขา เป็นพอร์ตแบบสองทิศทาง ใช้ในการรับส่งข้อมูล นอกจากนี้ยังทำหน้าที่อื่นๆ แสดงดังตารางที่ 2.5

ตารางที่ 2.5 หน้าที่ของขาสัญญาณของพอร์ต E

พอร์ต	สัญญาณ	หน้าที่
RE0	AN5	รับสัญญาณอินพุตสำหรับ ADC ช่อง 5
	RD	ขาสัญญาณขยายพอร์ตแบบขนานควบคุมการอ่าน
RE1	AN6	รับสัญญาณอินพุตสำหรับ ADC ช่อง 6
	WR	ขาสัญญาณขยายพอร์ตแบบขนานควบคุมการเขียน
RE2	AN7	รับสัญญาณอินพุตสำหรับ ADC ช่อง 7
	CS	ขาสัญญาณขยายพอร์ตแบบขนานควบคุมการเลือกอุปกรณ์

2.2 โมดูล RN-171

โมดูล RN 171 เป็นระบบปฏิบัติการเดี่ยวแบบสมบรูณ์ TCP/IP และเป็นโมดูลเครือข่ายไร้สาย เนื่องด้วยปัจจัยรูปแบบขนาดเล็กและการใช้พลังงานต่ำมาก RN-171 จึงเหมาะสำหรับการใช้งานแบบไร้สายโทรศัพท์มือถือ เช่น การตรวจสอบสินทรัพย์เซ็นเซอร์ และแบตเตอรี่แบบพกพา อุปกรณ์ที่ดำเนินการ และรวมถึง วิทยุ 2.4GHz, 32 -บิต SPARC processor, TCP / IP stack, นาฬิกาแบบเรียลไทม์ การเร่งเข้ารหัสลับ การจัดการพลังงานและการเชื่อมต่อเครื่องส่งสัญญาณโดยตรง โมดูลนี้จะโหลดไว้กับซอฟต์แวร์ที่บรรจุไว้ในชิป เพื่อให้ง่ายต่อการรวมและลดการพัฒนาโปรแกรมประยุกต์ของผู้ใช้งาน ในการตั้งค่าฮาร์ดแวร์ที่ง่ายที่สุดต้องการการเชื่อมต่อเพียง 4 อย่าง (PWR, TX, RX and GND) เพื่อสร้างการเชื่อมต่อข้อมูลแบบไร้สาย นอกจากนี้การนำเข้าแอนะล็อกเซ็นเซอร์สามารถใช้ในการเชื่อมต่อที่หลากหลายของเซ็นเซอร์ เช่น อุณหภูมิ เสียง การเคลื่อนไหว และ การเร่งความเร็ว ความสามารถในการเข้าสู่โหมดประหยัดพลังงานและสแกนอัตโนมัติและเชื่อมโยงไปยัง AP เมื่อกระตุ้น ทำให้ RN-171 เหมาะสำหรับการ roaming โปรแกรม RN-171 ยังรวมถึงการสร้างขึ้นในโปรแกรม HTML เพื่อแจ้งข้อมูลอัตโนมัติ serial uart หรือข้อมูล sensor ไปยังเว็บเซิร์ฟเวอร์

รูปที่ 2.2 โมดูล wifly RN-171

2.2.1 คุณสมบัติ

- FCC / CE / IC ได้รับการรับรอง 2.4GHz IEEE 802.11b / g transceiver
- มีขนาดเล็ก : 1050 x 700 x 130 mil
- กำลังส่งกำหนด: 0dBm ถึง 10 dBm
- แผ่น RF ต่อขั้วเสาอากาศ
- เสาอากาศได้รับการรับรอง: เสาอากาศ Chip, 4 "เสาอากาศสองแฉก , PCB และลวดเสาอากาศ
- พลังงานต่ำพิเศษ - 4uA, 38mA Rx, 120mA Tx ที่ 0dBm
- อัตราสูง - 921Kbps TX, RX 500Kbps อัตราการส่งข้อมูล TCP / IP และ WPA2 มากกว่า UART,ไม่เกิน 2Mbps SPI slave
- 8 Mbit หน่วยความจำแฟลชและ 128 KB สำหรับRAM
- 10 วัตถุประสงค์สำหรับ ดิจิตอล I / O
- 8 อินเตอร์เฟซเซ็นเซอร์อนาล็อก
- นาฬิกาเวลาจริงสำหรับการปลุกและเวลาปั๊ม
- ยอมรับไฟ 3.3V จากแหล่งจ่ายไฟ หรือ 3V จากแบตเตอรี่
- รองรับเครือข่ายและโครงสร้างพื้นฐาน Adhoc
- บนบอร์ดสมบูรณ์เครือข่าย TCP / IP stack
- เป็นมิตรกับสิ่งแวดล้อม ตามมาตรฐาน RoHS

2.2.2 การประยุกต์ใช้งาน

- ตรวจสอบระยะไกล
- เซนเซอร์ในอุตสาหกรรมและการควบคุม
- Telemetry (ระบบทำงานร่วมกันของ computer และ wireless network เพื่อนำมาซึ่งประโยชน์ในด้านต่างๆ)
- การทำงานหน้าแรกอัตโนมัติ

รายละเอียดการทำงานของแต่ละขาตามตารางที่ 2.6

ตารางที่ 2.6 การทำงานแต่ละขาของโมดูล RN-171

ขาที่	ชื่อสัญญาณ	รายละเอียด	ฟังก์ชันเสริม	ทิศทาง
1	GND	กราวน์		
2	Not Used	ไม่ต้องเชื่อมต่อ		ไม่เชื่อมต่อ
3	Not Used	ไม่ต้องเชื่อมต่อ		ไม่เชื่อมต่อ
4	GPIO 9	เปิดใช้งานโหมด Ad Hoc, คืนค่าเดิมจากโรงงาน, 8mA Drive, ทนไฟ 3.3V		อินพุต/เอาท์พุต
5	GPIO 8	GPIO, 24mA Drive, ทนไฟ 3.3V		อินพุต/เอาท์พุต
6	GPIO 7	GPIO, 24mA Drive, ทนไฟ 3.3V		อินพุต/เอาท์พุต

7	GPIO 6	GPIO, 24mA drive, ทนไฟ 3.3V, สถานะเชื่อมต่อเมื่อ Roving Firmware	สัมพันธ์กับสถานะ AP	อินพุต/เอาท์พุต
8	GPIO 5	GPIO, 24mA Drive, ทนไฟ 3.3V	สถานะข้อมูล Tx/Rx	อินพุต/เอาท์พุต
9	GPIO 4	GPIO, 24mA Drive, ทนไฟ 3.3V	เชื่อมต่อผ่านสถานะ TCP	อินพุต/เอาท์พุต
10	VDD_3.3V	ไฟเลี้ยง 3.3 V		
11	GPIO 3	GPIO, 8mA Drive, ทนไฟ 3.3V		อินพุต/เอาท์พุต
12	GPIO 2	GPIO, 8mA Drive, ทนไฟ 3.3V		อินพุต/เอาท์พุต
13	GPIO 1	GPIO, 8mA Drive, ทนไฟ 3.3V		อินพุต/เอาท์พุต
14	GND	กราวน์		
15	Not Used	ไม่ต้องเชื่อมต่อ		ไม่เชื่อมต่อ
16	Not Used	ไม่ต้องเชื่อมต่อ		ไม่เชื่อมต่อ
17	Not Used	ไม่ต้องเชื่อมต่อ		ไม่เชื่อมต่อ
18	Not Used	ไม่ต้องเชื่อมต่อ		ไม่เชื่อมต่อ
19	Not Used	ไม่ต้องเชื่อมต่อ		ไม่เชื่อมต่อ
20	GND	กราวน์		

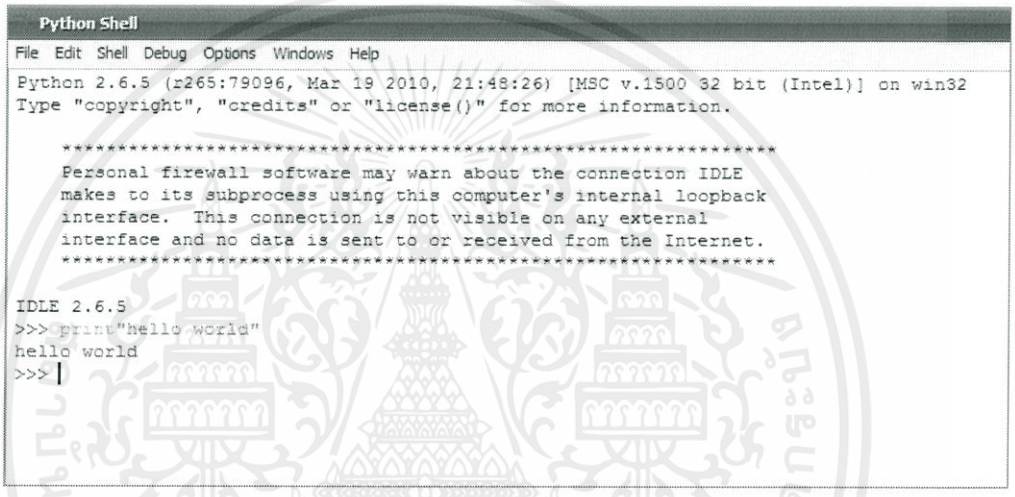
21	GND	กราวน์		
22	GND	กราวน์		
23	GND	กราวน์		
24	ANTENNA	802.11 b/g เสาอากาศ 2.4Ghz		
25	GND	กราวน์		
26	GND	กราวน์		
27	GND	กราวน์		
28	GND	กราวน์		
29	SENSOR 0	อินเตอร์เฟซเซนเซอร์, แอนะล็อกอินพุตเข้าโมดูล, ทนไฟ 1.2V		อินพุต
30	SENSOR 1	อินเตอร์เฟซเซนเซอร์, แอนะล็อกอินพุตเข้าโมดูล, ทนไฟ 1.2V		อินพุต
31	SENSOR 2	อินเตอร์เฟซเซนเซอร์, แอนะล็อกอินพุตเข้าโมดูล, ทนไฟ 1.2V		อินพุต
32	SENSOR 3	อินเตอร์เฟซเซนเซอร์, แอนะล็อกอินพุตเข้าโมดูล, ทนไฟ 1.2V		อินพุต
33	SENSOR POWER	เอาท์พุตจากโมดูลสูงสุด 3.3V		

34	VDD_3.3_RF	ไฟเลี้ยง RF 3.3V		
35	SENSOR 4	อินเตอร์เฟซเซนเซอร์, แอนะล็อกอินพุตเข้าโมดูล, ทนไฟ 1.2V		อินพุต
36	SENSOR 5	อินเตอร์เฟซเซนเซอร์, แอนะล็อกอินพุตเข้าโมดูล, ทนไฟ 1.2V		อินพุต
37	SENSOR 6	อินเตอร์เฟซเซนเซอร์, แอนะล็อกอินพุตเข้าโมดูล, ทนไฟ 1.2V		อินพุต
38	SENSOR 7	อินเตอร์เฟซเซนเซอร์, แอนะล็อกอินพุตเข้าโมดูล, ทนไฟ 1.2V		อินพุต
39	GND	กราวน์		
40	RESET	รีเซ็ตสัญญาณ(Active Low)ใช้พัลส์อย่างน้อย 160us		อินพุต
41	FORCE_AWAKE	กระตุ้นสัญญาณ(Active High) ใช้พัลส์อย่างน้อย 260us		อินพุต
42	GPIO 14	GPIO, 8mA Drive, ทนไฟ 3.3V		อินพุต/เอาท์พุต
43	UART_RTS	ควบคุมการไหลของ UART RTS, กระแส 8mA , ทนไฟ 3.3V		เอาท์พุต

44	UART_CTX	ควบคุมการไหลของ UART CTX, ทนไฟ 3.3V		อินพุต
45	UART_RX	UART TX, ทนไฟ 3.3V		อินพุต
46	UART_TX	UART TX, กระแส 8mA , ทนไฟ 3.3V		เอาท์พุต
47	GND	กราวน์		
48	SREG_3V3_CTRL	เพิ่มการควบคุม Regulator		เอาท์พุต
49	VDD-BAT	ใส่แบตเตอรี่ 2.0-3.3V กับ Regulator เพิ่มในการใช้งาน, เชื่อมต่อกับ VDD ถ้าไม่ได้ใช้ Regulator		

2.3 ภาษาไพธอน

ต้นกำเนิดของภาษาไพธอนถูกสร้างขึ้นครั้งแรกเมื่อปีพ.ศ.2533 โดย Mr. Guido Van Rossum ในรุ่น 0.9.0 ภาษาไพธอน คือภาษาที่ใช้ในการเขียนโปรแกรมภาษาหนึ่งที่มีความสามารถสูงไม่แพ้ภาษาอื่นๆที่มีอยู่ในปัจจุบัน ภาษาไพธอนนั้นเป็นภาษาอิสระ (Open Source) ทำให้ทุกคนสามารถที่จะนำภาษาไพธอนมาพัฒนาโปรแกรมโดยไม่ต้องเสียค่าใช้จ่าย ทำให้มีคนเข้ามาช่วยกันพัฒนาให้ภาษาไพธอนมีความสามารถสูงขึ้น และใช้งานได้ครอบคลุมกับทุกลักษณะงาน



รูป 2.3 หน้าต่างโปรแกรมไพธอน (Python)

ภาษาไพธอนนั้นมีเครื่องมือสำเร็จรูปต่างๆให้เลือกใช้มากมาย เช่น เครื่องมือที่ทำงานด้านการประมวลผลภาพ เครื่องมือที่ใช้ในการรับและส่งค่าผ่านทางพอร์ตอนุกรม เครื่องมือที่ใช้ในการแสดงผลภาพจากกล้องดิจิตอล ด้วยเหตุผลทั้งหมดภาษาไพธอนจึงเป็นภาษาที่น่าสนใจและถูกนำมาใช้ประโยชน์ในโครงงานนี้

2.4 การประมวลผลภาพดิจิตอล

การมองเห็นของมนุษย์เป็นสิ่งสำคัญและเป็นกลไกการรับภาพที่ซับซ้อนอย่างหนึ่ง ซึ่งจะให้ข้อมูลที่มีความจำเป็นสำหรับใช้ในงานง่ายๆ (ตัวอย่างเช่น การจดจำวัตถุ) และสำหรับงานที่มีความซับซ้อน (ได้แก่ การวางแผน การตัดสินใจ การค้นคว้าทางวิทยาศาสตร์ การพัฒนาทางด้านความคิด) รูปภาพมีบทบาทมากสำหรับองค์กรต่างๆ เช่น หนังสือพิมพ์ โทรทัศน์ ภาพยนตร์ซึ่งได้ใช้ภาพ (ภาพนิ่ง ภาพเคลื่อนที่) เป็นสื่อนำเสนอข้อมูลข่าวสารต่าง ๆ สิ่งที่น่าสนใจของข้อมูลที่เกี่ยวกับการมองเห็นหรือข้อมูลภาพนั้นก็คือกระบวนการประมวลผลภาพ (Image Processing) โดยใช้ดิจิตอลคอมพิวเตอร์

ความพยายามทางด้านการประมวลผลภาพได้เริ่มขึ้นในปี 1964 ณ ห้องแลป Jet Propulsion (Pasasena California) ซึ่งได้นำกระบวนการการประมวลผลภาพมาใช้ในการพิจารณาภาพถ่ายดาวเทียมของดวงจันทร์ต่อมาได้มีการตั้งสาขาทางวิทยาศาสตร์สาขาใหม่มีชื่อว่า Digital image processing หลังจากนั้นงานทางด้านการประมวลผลภาพก็พัฒนาขึ้นเรื่อย ๆ และใช้กันอย่างกว้างขวางสำหรับงานในหลายๆ ด้าน ตัวอย่างเช่น ทางด้านสื่อสารโทรคมนาคม การสื่อสารทางโทรทัศน์ทางด้านการพิมพ์ทางด้านกราฟฟิกการแพทย์และการค้นคว้าทางวิทยาศาสตร์

Digital image processing จะเกี่ยวกับการแปลงข้อมูลภาพให้อยู่ในรูปแบบข้อมูลดิจิตอล (Digital format) ซึ่งสามารถที่จะนำเอาข้อมูลนี้จัดผ่านกระบวนการต่าง ๆ ด้วยดิจิตอลคอมพิวเตอร์ได้ในระบบของดิจิตอล อินพุตและเอาพุตของระบบจะอยู่ในรูปแบบดิจิตอลเท่านั้น

Digital image analysis จะเกี่ยวกับวิธีการอธิบายและการจดจำข้อมูลภาพดิจิตอล ซึ่งอินพุตของระบบจะเป็นข้อมูลภาพดิจิตอลและเอาพุตจะเป็นเครื่องหมายที่ใช้แทนข้อมูลภาพดิจิตอลเหล่านั้น ในการวิเคราะห์ภาพมีอยู่หลายวิธีด้วยกันที่ได้นำมาจากการทำงานของตามนุษย์ (human vision) นั่นก็คืองานทางด้าน Computer Vision เป็นลักษณะเดียวกับ Digital image analysis นั่นเอง การมองเห็นของมนุษย์นับว่าเป็นกระบวนการที่ซับซ้อนซึ่งลักษณะเทคนิคโดยทั่วไปในกระบวนการ Digital image analysis และ Computer Vision จะค่อนข้างซับซ้อนเช่นกัน

2.4.1 รูปร่างของภาพ (Image Shape)

วัตถุที่มีอยู่ตามธรรมชาติและที่มนุษย์สร้างขึ้นมีรูปร่างที่แตกต่างกันไปทั้งที่เป็นรูปทรงเรขาคณิตและไม่เป็นรูปทรงเรขาคณิต ในศาสตร์ของการประมวลผลภาพนั้นการกำหนดขอบเขตของภาพทุกภาพให้อยู่ในรูปสี่เหลี่ยม (Rectangular image model) เป็นวิธีที่นิยมใช้กันมากที่สุด เนื่องจากทำให้การอ่านภาพ การจัดเก็บข้อมูลภาพในหน่วยความจำ และการแสดงภาพออกทางอุปกรณ์ต่างๆเป็นไปได้อย่างมีประสิทธิภาพ

การเก็บข้อมูลภาพลงหน่วยความจำของคอมพิวเตอร์สามารถทำได้โดยการจองหน่วยความจำของเครื่องไว้ในรูปของตัวแปรอะเรย์ (array) โดยค่าในแต่ละช่องของอะเรย์แสดงถึงคุณสมบัติของจุดภาพ (pixel) และตำแหน่งของช่อง อะเรย์เป็นตัวกำหนดตำแหน่งของจุดภาพ

สมมุติให้ Image เป็นตัวแปรแบบอะเรย์ขนาด M X N (M แถวและ N คอลัมน์) ที่ใช้เก็บภาพขนาด M x N จุด (M จุดในแนวนอน และ N จุดในแนวตั้ง) ค่าสี (หรือความสว่าง ในกรณีที่เป็นภาพ grey level) ของจุดภาพในแถวที่ 5 คอลัมน์ที่ 4 จะตรงกับค่าของ Image(5,4) จะเห็นว่าเราใช้ตำแหน่งของจุดภาพทั้งสองแกนเป็นตัวชี้ค่าข้อมูลในอะเรย์

จากการใช้หน่วยความจำเพื่อการเก็บภาพในลักษณะที่กล่าวมา เนื้อที่ในการเก็บภาพสามารถคำนวณได้จาก M x N x g เมื่อ g เป็นจำนวนเต็มที่แทนจำนวนบิตของข้อมูลในแต่ละจุดภาพ ตัวอย่างถ้า g มีค่าเท่ากับ 8 บิต เราจะสามารถเก็บความแตกต่างของระดับสีที่เป็นไปสูงสุด 256 ระดับ ค่า M และ N จะเป็นตัวบอกถึงความละเอียดของภาพ สำหรับคอมพิวเตอร์ทั่วไปในระบบ VGA (Video Graphic Array) จะมีขนาด 640x480, 800x600 และ 1024x768 จุด เป็นต้น การกำหนดความละเอียดจะขึ้นอยู่กับงานที่จะใช้ในงานบางอย่างใช้ความละเอียดแค่ 30 x 50 จุด ก็พอแล้ว แต่ในงานบางชนิด ใช้ความละเอียดถึง 1000 x 1000 จุด ก็ยังไม่พอ

ปกติแล้วในการเก็บข้อมูลภาพโดยเครื่องมือต่างๆจะเก็บตามมาตรฐานของโทรทัศน์ซึ่งมีอัตราส่วน x ต่อ y เท่ากับ 4:3 สำหรับเครื่องมือเก็บข้อมูลภาพที่ไม่เป็นไปตามอัตราส่วน 4:3 เมื่อนำภาพนี้ไปแสดงในจอภาพมาตรฐานจะทำให้ภาพที่แสดงนั้นมีขนาดของจุดภาพไม่เป็นสี่เหลี่ยมจัตุรัส เช่น ในบางระบบอาจจะใช้ความละเอียดในการแสดงเท่ากับ 640 x 512 ซึ่งจะทำให้ขนาดของจุดภาพที่ได้มีขนาดของด้านกว้างมีความยาวมากกว่าด้านสูง ซึ่งลักษณะดังกล่าวนี้

เป็นหัวข้อที่ต้องสนใจสำหรับการเขียนโปรแกรมทางด้านกราฟฟิกและการจัดการข้อมูลจำนวนสีสูงสุดที่เป็นไปได้ของแต่ละจุดภาพ ขึ้นอยู่กับจำนวนบิตที่ใช้ เมื่อมีการกำหนดให้ขนาดของบิตต่อจุดมากขึ้นจะทำให้จำนวนของสีมากขึ้นด้วย ตัวอย่างเช่น

1 บิต = 2^1 = 2 สี

2 บิต = 2^2 = 4 สี

4 บิต = 2^4 = 16 สี

8 บิต = 2^8 = 256 สี

16 บิต = 2^{16} = 65536 สี เป็นต้น

สำหรับการแสดงข้อมูลภาพที่มีขนาด 1 บิตและ 8 บิตนั้นจะมีการทำงานที่จะใกล้เคียงกันเนื่องจากหน่วยประมวลผลจะไม่สามารถจัดการกับข้อมูลที่เป็นบิตเดี่ยว ๆ ได้ดังนั้นในการแสดงข้อมูลออกทางจอภาพตัวโปรเซสเซอร์จะทำการก็อปปี้ข้อมูลทั้ง 8 บิต (1 Byte) ส่งให้กับจอภาพซึ่งในกรณีที่ Pixel มีขนาด 1 บิต เมื่อโปรเซสเซอร์จะทำงานกับบิตแรกที่ต้องการแล้วก็จะทำการก็อปปี้ข้อมูลชุดใหม่ทันทีโดยที่ไม่เกี่ยวกับข้อมูลอีก 7 บิตที่เหลือส่วนในกรณี Pixel ที่มีขนาด 8 บิต โปรเซสเซอร์จะทำการก็อปปี้ข้อมูลจุดใหม่ก็ต่อเมื่อโปรเซสเซอร์ทำงานกับทุกบิตแล้ว

ตัวอย่างสำหรับระบบที่มีความละเอียดเท่ากับ 800x600 และมีขนาด 16 บิตต่อ Pixel จะสามารถแสดงสีได้ทั้งหมด 65536 ระดับและต้องใช้เนื้อที่ในการเก็บเท่ากับ 800x600x16 บิต

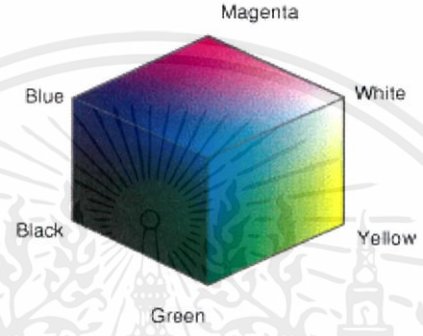
2.4.2 มาตรฐานของสี

มาตรฐานของสีที่ใช้อยู่ในปัจจุบันมีอยู่หลายระบบด้วยกัน ทั้งนี้จะขึ้นอยู่กับการนำไปใช้ แต่โดยทั่วไปแล้วทุกมาตรฐานจะมีแนวคิดเดียวกันคือ การแทนจุดสีด้วยจุดที่อยู่ภายในสเปซ 3 มิติ โดยจะมีแกนอ้างอิงสำหรับจุดสีนั้นในสเปซซึ่งแต่ละแกนจะมีความเป็นอิสระต่อกัน ตัวอย่างเช่นในระบบ RGB จะมีแกนสีคือ แกนสีแดง เขียว และน้ำเงินในระบบ HSV จะมีแกนเป็นค่าสี(Hue) ความอิ่มตัวของสี(Saturation) และค่าความสว่างของแสง (Value)

ตัวอย่างระบบสีที่นิยมใช้กันได้แก่ ระบบ RGB และ HSV (Hue Saturation Value)

2.4.2.1 ระบบสี RBG

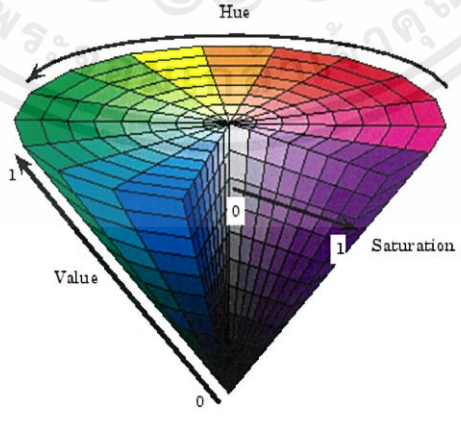
ระบบสี RGB เป็นปริภูมิสีที่ใช้กันอย่างแพร่หลายในระบบภาพดิจิตอล คอมพิวเตอร์ โดยแทนค่าสีของจุดภาพแต่ละจุดภาพด้วยเวกเตอร์สามมิติ ซึ่งแทนค่าสีปฐมภูมิได้แก่ สีแดง, สีเขียว และ สีน้ำเงิน โดยช่วงค่าของแต่ละสีอยู่ระหว่าง 0-255 ซึ่งปริภูมิสี RGB จะมีลักษณะเป็นรูปลูกบาศก์ ดังรูป



รูปที่ 2.4 ปริภูมิสี RGB

2.4.2.2 ระบบสี HSV

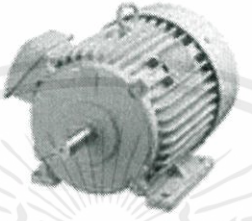
ระบบสี HSV ใช้หลักการแยกความสว่างออกจากเนื้อสีของจุดภาพ โดยปริภูมิสี HSV นั้นแทนค่าด้วยเวกเตอร์สามมิติ ซึ่งประกอบด้วย H แทนค่าเนื้อสี (Hue), S แทนค่าความอิ่มตัวของสี (Saturation) และ V แทนค่าความสว่างของแสง (Value) ซึ่งปริภูมิสี HSV จะมีลักษณะเป็นรูปกรวย ดังรูป



รูป 2.5 ปริภูมิสี HSV

2.5 มอเตอร์กระแสตรง (DC Motor)

เครื่องใช้ไฟฟ้าประเภทเครื่องกลที่ใช้งานกันอยู่ทั่วไปภายในอาคารบ้านเรือนและโรงงานอุตสาหกรรม ได้แก่ พัดลม เครื่องบด เครื่องปั่น เครื่องซักผ้า ปั๊มน้ำ เครื่องกลึง เครื่องไส เครื่องเจาะ เครื่องคว้าน เครื่องเจียระไน เป็นต้น ต่างก็ทำงานด้วยการหมุนขับของมอเตอร์จึงเป็นเครื่องกลไฟฟ้าที่ให้กำเนิดพลังงานที่จำเป็นและสำคัญยิ่งประเภทหนึ่ง



รูปที่ 2.6 มอเตอร์กระแสตรง

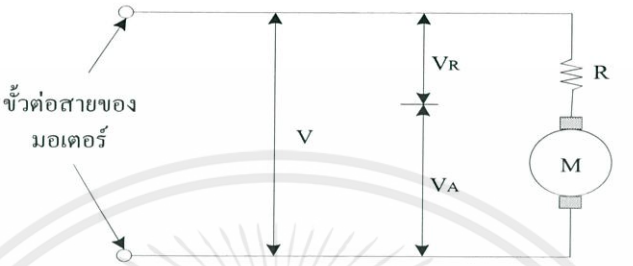
2.5.1 หลักการทำงานของมอเตอร์กระแสตรง

เมื่อมีการผ่านกระแสไฟฟ้าเข้าไปยังขดลวดในสนามแม่เหล็กจะทำให้เกิดแรงแม่เหล็กซึ่งมีสัดส่วนของแรงขึ้นกับกระแสแรงของสนามแม่เหล็ก โดยแรงจะเกิดขึ้นเป็นมุมฉากกับกระแสและสนามแม่เหล็ก ขณะที่ทิศทางของแรงดันกลับตรงกันข้ามกัน ถ้าหากกระแสของสนามแม่เหล็กไหลย้อนกลับจะทำให้เกิดการเปลี่ยนแปลงของกระแสและสนามแม่เหล็กเป็นผลทำให้ทิศทางของแรงเปลี่ยนไป ด้วยคุณสมบัตินี้ทำให้มอเตอร์กระแสตรงกลับทิศทางการหมุนได้

สนามแม่เหล็กของมอเตอร์ส่วนหนึ่งเกิดขึ้นจากแม่เหล็กถาวรซึ่งจะถูกยึดติดกับแผ่นเหล็ก หรือ เหล็กกล้า โดยปกติส่วนนี้จะเป็นส่วนที่ยึดอยู่กับที่ และขดลวดเหนี่ยวนำจะพันอยู่กับส่วนที่เป็นแกนหมุนของมอเตอร์

2.5.2 คุณสมบัติของมอเตอร์กระแสตรง

ในการอธิบายคุณสมบัติของมอเตอร์กระแสตรงให้ละเอียดนั้นต้องพิจารณาแรงดันที่ป้อนและความต้านทานของโรเตอร์ด้วย วงจรภายในของมอเตอร์เขียนได้ดังรูปที่ 2.7



รูปที่ 2.7 วงจรภายในของมอเตอร์กระแสตรง

โดยสมมติให้ทุ่นโรเตอร์ไม่มีความต้านทานอยู่เลย อนุกรมกับความต้านทานซึ่งในที่นี้คือความต้านทานของขดลวดนั่นเอง แรงดันที่ขั้วต่อสายของมอเตอร์ก็คือผลบวกระหว่างแรงดันที่ทุ่นโรเตอร์ (V_A) และแรงดันตกคร่อมความต้านทานขดลวด (V_R) แรงดัน V_A ถูกเรียกว่า แรงเคลื่อนเหนี่ยวนำป้อนกลับ (BACK EMF) ซึ่งเกิดขึ้นในโรเตอร์ขณะที่หมุนแรงดันที่เกิดขึ้นนี้เป็นไปตามกฎของการเหนี่ยวนำแม่เหล็กไฟฟ้าจากการเคลื่อนที่ของตัวนำในสนามแม่เหล็กสัมพันธ์กับแรงเคลื่อนเหนี่ยวนำแม่เหล็กและความเร็วในการเคลื่อนที่ของตัวนำ แรงดันที่เกิดขึ้นจะมีขั้วตรงข้ามกับแรงดันที่ป้อนให้กับมอเตอร์ และแปรผันตรงกับความเร็วในการหมุน ผลบวกของแรงดันที่ทุ่นโรเตอร์ (V_A) และแรงดันตกคร่อมขดลวด (V_R) ต้องเท่ากับแรงดันที่ป้อนให้กับมอเตอร์ (V)

$$V = V_A + V_R \quad (V) \qquad \text{(สมการที่ 2.1)}$$

เมื่อพิจารณาตั้งแต่มอเตอร์หยุดนิ่ง ความเร็วมีค่าเป็นศูนย์ ดังนั้น $V_A = 0$, $V_R = V$ กระแสที่ไหลในมอเตอร์หาได้จาก

$$I = V_R / R \quad (A) \qquad \text{(สมการที่ 2.2)}$$

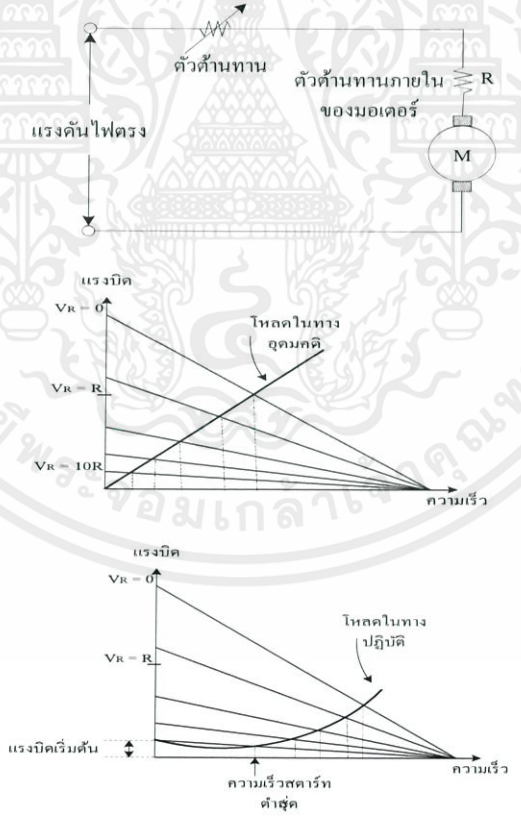
เมื่อมอเตอร์เริ่มหมุนจะมีความเร็ว และ V_A เพิ่มขึ้นเป็นเส้นตรงตามความเร็ว V_R ซึ่งมีค่าเท่ากับความแตกต่างระหว่าง V_A และ V จะเริ่มลดลง กระแส I ก็จะเริ่มลดลงเช่นกันขณะที่มอเตอร์ยังมีความเร่งอยู่ ความเร็วจะเพิ่มขึ้น แรงบิดจะลดลงจนกว่าจะถึงจุดซึ่งแรงบิดของมอเตอร์

รับภาระโหลดได้สมดุลพอดี ขณะที่มอเตอร์ไม่มีโหลดและหมุนอย่างอิสระจะมีเพียงค่าความฝืดของแบริ่ง และแรงต้านอากาศทำให้ V_A เกือบเท่ากับค่า V

2.5.3 การควบคุมความเร็วของมอเตอร์ขั้นพื้นฐาน

2.5.3.1 การควบคุมด้วยตัวต้านทานที่ปรับค่าได้

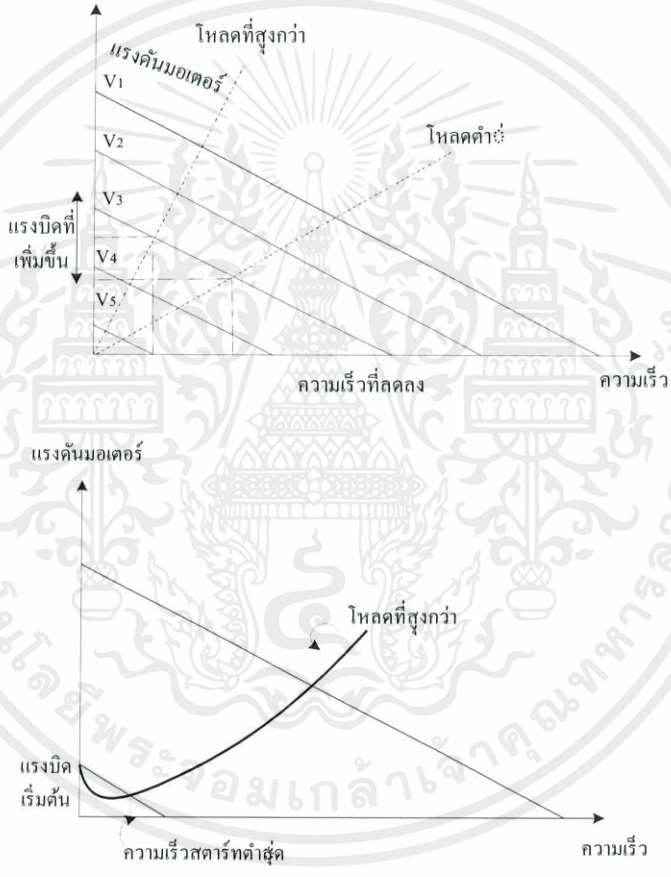
เป็นรูปแบบพื้นฐานที่สุดของการควบคุมมอเตอร์คือ ใช้ตัวต้านทานปรับค่าได้อนุกรมกับมอเตอร์ โดยตัวต้านทานที่ปรับค่าได้จะเป็นตัวกำหนดความเร็วในการหมุนของมอเตอร์ การบังคับแบบนี้ไม่มีประสิทธิภาพเพราะกำลังไฟสูญเสียไปในความต้านทาน มักนิยมใช้กับมอเตอร์ตัวเล็กๆ การบังคับแบบนี้ให้คุณสมบัติการสตาร์ทดี(ให้แรงบิดสูงที่ความเร็วต่ำ)แต่จะให้ความเร็วสูงมากเมื่อมอเตอร์อยู่ในภาวะที่มีโหลดน้อยๆ ดังนั้นการบังคับแบบนี้มีประโยชน์เฉพาะภาวะที่แรงต้านคงที่ เช่น การบังคับความเร็วของเครื่องจักรเย็บผ้า เป็นต้น



รูปที่ 2.8 วงจรควบคุมความเร็วของมอเตอร์กระแสตรงแบบใช้ตัวต้านทานอนุกรมและกราฟคุณสมบัติ

2.5.3.2 การควบคุมด้วยวิธีเปลี่ยนค่าแรงดัน

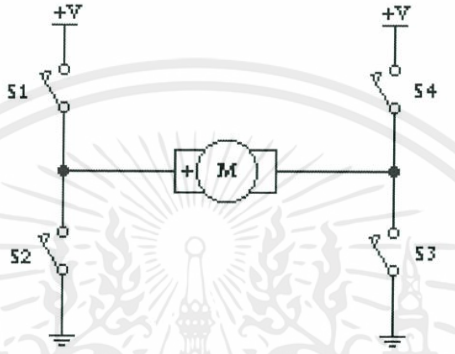
วิธีการนี้ดีกว่าวิธีการแรกแต่จะซับซ้อนกว่า ต้องใช้อุปกรณ์อิเล็กทรอนิกส์ที่มีอัตราขยายกำลังสูง และมอเตอร์จะถูกป้อนด้วยแรงดันที่เปลี่ยนแปลงค่าได้ จากแหล่งจ่ายที่มีอิมพีแดนซ์ต่ำ ข้อดีของการควบคุมวิธีนี้คือ ถ้าความเร็วลงลดจากผลของแรงบิด แรงดันที่ป้อนให้กับมอเตอร์จะเพิ่มขึ้นเพื่อรักษาระดับความเร็ว ส่วนข้อเสียจากการควบคุมวิธีนี้คือ เมื่อมอเตอร์มีความเร็วต่ำแรงดันที่ป้อนให้กับมอเตอร์จะมีค่าต่ำเช่นกัน



รูปที่ 2.9 การควบคุมความเร็วโดยเปลี่ยนค่าแรงดัน

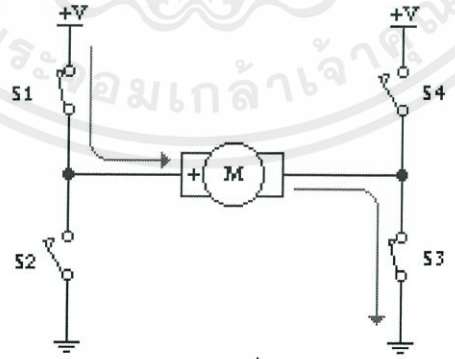
2.5.4 ทฤษฎีการควบคุมทิศทางการหมุนของมอเตอร์

การควบคุมทิศทางการหมุนของมอเตอร์ให้หมุนได้ทั้งเดินหน้าและถอยหลังนั้น สามารถทำได้หลายวิธีด้วยกัน ซึ่งนิยมใช้วงจร H-Bridge Switching จากหลักการของวงจรนั้นจะประกอบไปด้วย สวิตช์ 4 ตัว นั่นก็คือ S1,S2,S3 และ S4 นั่นเอง ซึ่งในรูปตัวอย่าง จะใช้ DC-Motor เป็น Load ของวงจรนั่นเอง



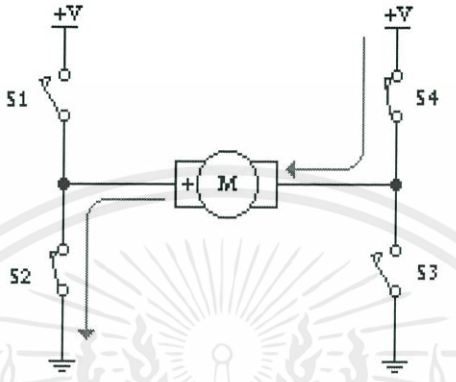
รูปที่ 2.10 วงจร H-Bridge Switching

ในสภาวะเริ่มต้น สวิตช์ทุกตัว Off อยู่ก็จะไม่มีอะไรเกิดขึ้นทั้งสิ้น เพราะไม่มีกระแสไฟฟ้าไหลเข้าสู่มอเตอร์ และเมื่อเราทำการ On สวิตช์ S1 และ S3 พร้อมกัน จะเป็นการเชื่อมวงจร ทำให้มีกระแสไฟฟ้าไหลผ่านมอเตอร์ จากขั้วบวกของมอเตอร์ ไปยังขั้วลบของมอเตอร์ จึงทำให้มอเตอร์สามารถหมุนได้ในทิศทาง Forward (จะหมุนแบบตามเข็มนาฬิกา หรือ ทวนเข็มนาฬิกานั้น ขึ้นอยู่กับการพันขดลวดภายในมอเตอร์)



รูปที่ 2.11 วงจร H-Bridge Switching ที่สวิตช์ S1 และ S3 On พร้อมกัน

และในทางกลับกัน ถ้าหากเราทำการ On สวิตซ์ S2 และ S4 พร้อมกัน ก็จะเป็นการเชื่อมวงจรและทำให้เกิดกระแสไฟฟ้าไหลผ่านมอเตอร์จากขั้วลบของมอเตอร์ ไปยังขั้วบวกของมอเตอร์ จึงทำให้มอเตอร์สามารถหมุนได้ และเป็นการหมุนในทิศทาง Reverse (กลับทิศทางกับกรณีแรก)



รูปที่ 2.12 วงจร H-Bridge Switching ที่สวิตซ์ S1 และ S4 On พร้อมกัน

ดังนั้นวงจรนี้จะอาศัยสวิตซ์ 4 ตัว เพื่อบังคับทิศทางการไหลของกระแสไฟฟ้าที่ไหลผ่านมอเตอร์ เพื่อควบคุมให้มอเตอร์หมุนตามทิศทางที่เราต้องการ โดยการผลัดกัน On และ Off สวิตซ์พร้อมกัน 2 ตัว

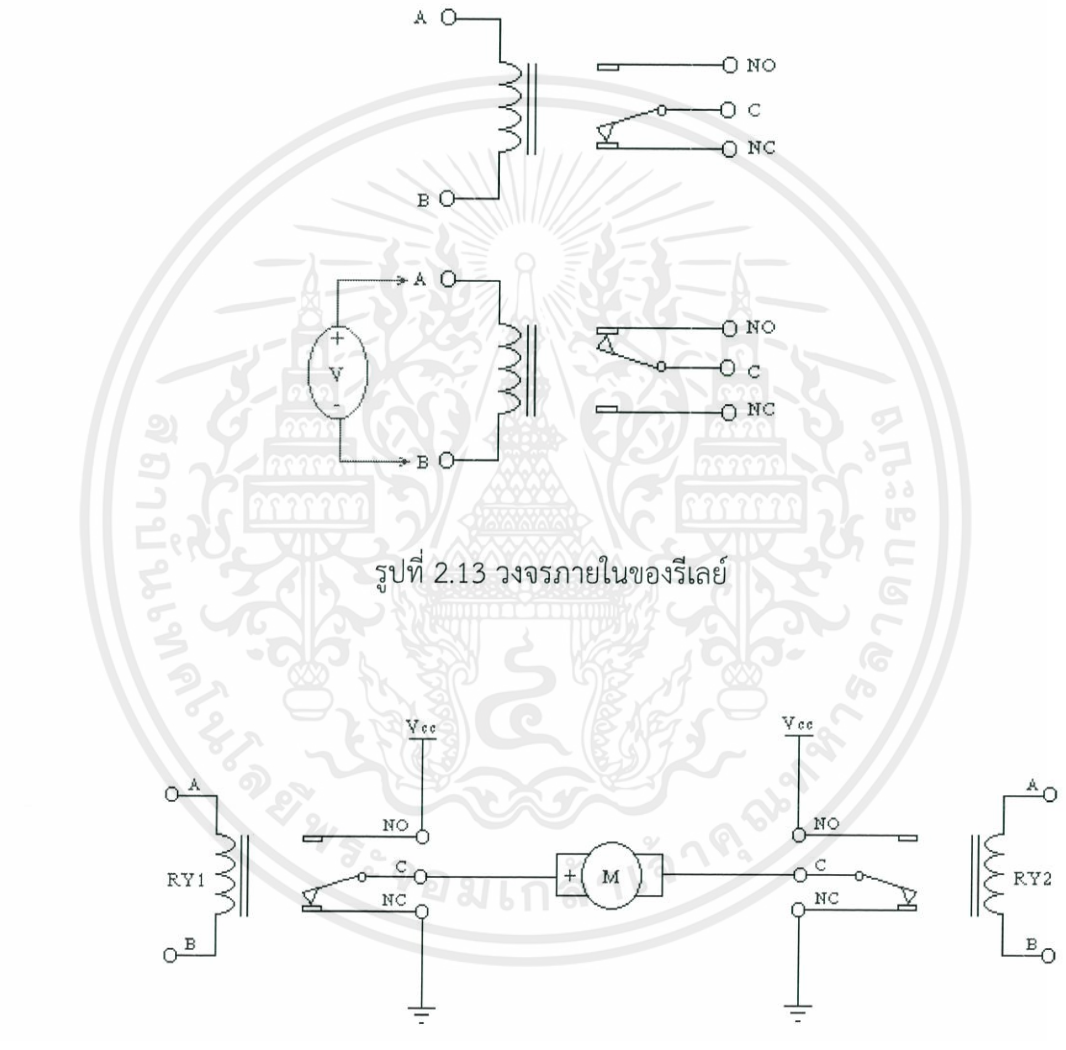
2.5.4.1 การสร้างวงจร H-Bridge Switching จาก Relay

รีเลย์เป็น อุปกรณ์แม่เหล็ก (Magnetic device) ซึ่งเป็นที่นิยมใช้กันมากในปัจจุบันรีเลย์ ถูกพัฒนาให้มีคุณภาพดีกว่าสมัยก่อนมาก แต่ยังคงหลักการและโครงสร้างเดิมเอาไว้ ภายในโครงสร้างของ รีเลย์ จะประกอบไปด้วยขดลวด (Coil) 1 ชุด และ หน้าสัมผัส (Contactor) ซึ่งในหน้าสัมผัส 1 ชุด จะประกอบไปด้วย

-หน้าสัมผัสแบบปกติปิด (Normally Close หรือ NC.) ซึ่งในสภาวะปกติขานี้จะต่ออยู่กับขาร่วม (Common)

-หน้าสัมผัสแบบปกติเปิด (Normally Open หรือ NO.) ขานี้จะต่อเข้ากับขาร่วม (Common) เมื่อ ขดลวดมีแรงดันตกคร่อม หรือกระแสไหลผ่าน (ในปริมาณที่เพียงพอ)

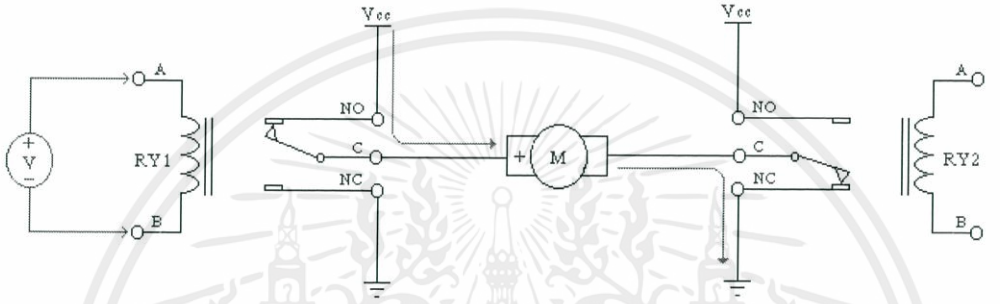
ใน รีเลย์ 1 ตัว อาจมีหน้าสัมผัสมากกว่า 1 ชุด เช่น 2 ชุด, 4 ชุด เป็นต้น ขึ้นอยู่กับผู้ผลิต เมื่อขดลวดได้รับแรงดันตกคร่อม (ขา A และ B) จะทำให้มีกระแสไหลผ่านขดลวด ซึ่งจะทำให้เกิด อำนาจสนามแม่เหล็ก ดึงดูดให้หน้าสัมผัส NO และ C ติดกัน ดังรูป



รูปที่ 2.13 วงจรภายในของรีเลย์

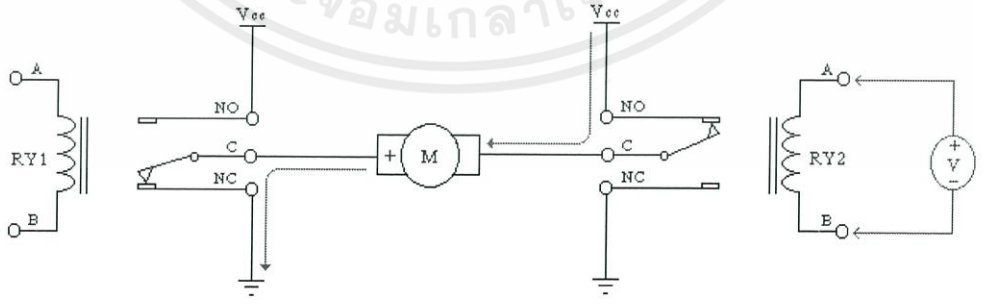
รูปที่ 2.14 วงจร H-Bridge Switching โดยใช้ Relay

วงจรนี้ จะประกอบไปด้วย รีเลย์ 2 ตัว คือ RY1 และ RY2 ซึ่ง Load คือ DC-Motor ซึ่งต่ออยู่กับ ขาร่วม (C) ของ RY1 และ RY2 โดยขั้วบวก (+) ของมอเตอร์ ต่ออยู่ที่ขา C ของ RY1 และขั้วลบ (-) ของมอเตอร์ ต่ออยู่ที่ขา C ของ RY2 โดยที่ขา NO ของ RY1 และ RY2 จะต่ออยู่กับขั้วบวกของแหล่งจ่ายไฟที่จะจ่ายให้มอเตอร์ (Vcc) และขา NC ของ RY1 และ RY2 จะต่อลงกราวด์



รูปที่ 2.15 กรณีที่ RY1 ทำงาน

เมื่อ RY1 ทำงาน (มีกระแสไหลผ่านขดลวดในปริมาณที่เพียงพอ) จะทำให้เกิดอำนาจสนามแม่เหล็กไฟฟ้าดึงดูดให้ขา NO และขา C ของ RY1 ติดกัน ส่งผลให้มีกระแสไหลจากแหล่งจ่าย (Vcc) ผ่านเข้าสู่ขั้วบวก (+) ของมอเตอร์ผ่านไปยังขา C ของ RY2 ซึ่งต่ออยู่ที่ NC และลงกราวด์ ทำให้มีกระแสไหล ผ่านมอเตอร์ในทิศทางบวกและครบวงจร จึงทำให้มอเตอร์สามารถหมุนในทิศทาง Forward ได้



รูปที่ 2.16 กรณี RY2 ทำงาน

เมื่อ RY2 ทำงาน (มีกระแสไหลผ่านขดลวดในปริมาณที่เพียงพอ) จะทำให้เกิดอำนาจ สนามแม่เหล็กไฟฟ้าดึงดูดให้ขา NO และขา C ของ RY2 ติดกัน ส่งผลให้มีกระแสไหลจากแหล่งจ่าย (Vcc) ผ่านเข้าสู่ขั้วลบ (-) ของมอเตอร์ผ่านไปยังขา C ของ RY1 ซึ่งต่ออยู่ที่ NC และลงกราวด์ ทำให้มีกระแสไหลผ่านมอเตอร์ในทิศทางลบและครบวงจร จึงทำให้มอเตอร์สามารถหมุนในทิศทาง Reward ได้

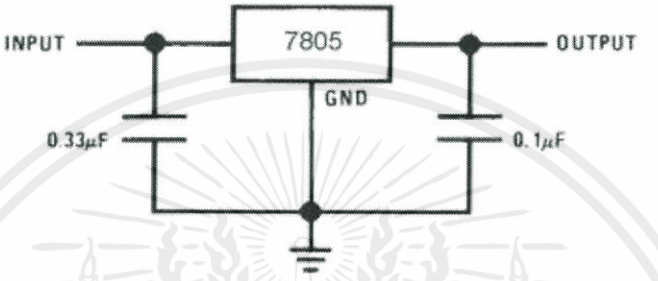
ผลกระทบของการตอบสนองของรีเลย์ เป็นอุปกรณ์ที่มีความเร็วในการทำงานต่ำ เช่น รีเลย์ชนิดแรงดันต่ำ (กระตุ้นขดลวดไม่เกิน 24v.) จะใช้เวลาในการทำงานประมาณ 10 - 50 mS. และรีเลยขนาด ใหญ่ที่ใช้ควบคุมมอเตอร์ในโรงงานอุตสาหกรรมนั้น อาจใช้เวลาในการทำงานมากกว่า 100 mS.

ผลกระทบจากอำนาจแม่เหล็กของรีเลย์ เป็นอุปกรณ์แม่เหล็ก ดังนั้นจึงไม่สามารถแก้ปัญหานี้ได้ ดังนั้นหลายคนอาจเคยมีปัญหาของรีเลย์ไปรบกวนการทำงานของวงจรไมโครคอนโทรลเลอร์ การแก้ไขมีหลายวิธี เช่น

- แยกกราวด์ : คือ การแยกกราวด์ของแหล่งจ่ายไฟสำหรับวงจรไมโครคอนโทรลเลอร์ และแหล่งจ่ายไฟกระตุ้น รีเลย์ ออกจากกันโดยใช้อุปกรณ์ Opto Cuppler
- แยกบอร์ด : คือ การแยกการทำงานในส่วนของวงจรรีเลย์ ออกไปจากบอร์ดไมโครคอนโทรลเลอร์แล้วทำการชีลด์ให้ดี
- ลดขนาดแรงดันกระตุ้น : คือ การเปลี่ยนตัวรีเลย์ เช่น เปลี่ยนจากรีเลย์ขนาด 24v. มาเป็นขนาด 12v.

2.6 วงจรรักษาแรงดันใช้ไอซีรักษาแรงดัน (IC Regulator)

ข้อดีของวงจรแบบนี้คือ ราคาถูก มีขนาดเล็ก และรูปแบบวงจรที่ง่าย สามารถจ่ายกระแสเอาท์พุตได้ตั้งแต่ 3 mA ถึง 100 mA ตามเบอร์ที่เราเลือกใช้ ยังมีวงจรป้องกันกระแสเกินภายในและวงจรป้องกันอุณหภูมิที่ทำงานให้เราเลือกตามต้องการวงจรรักษาแรงดันแบบ IC



รูปที่ 2.17 วงจรรักษาระดับแรงดันที่ใช้ไอซีเบอร์ LM7805

2.7 IP Camera Foscam FI8909W

รูปที่ 2.18 IP Camera Foscam FI8909W

Foscam FI8909W เป็นกล้องแบบ Wireless IP Camera ที่สามารถบันทึกได้ทั้งภาพและเสียง สามารีโมตเข้ามาดูภาพ มีความสามารถในการถ่ายภาพทั้งในเวลากลางวันและกลางคืน สามารถสร้างระแบบโครงข่ายในการบันทึกวิดีโอ สามารถใช้กับสมาร์ทโฟน ทั้ง (iOS, Andriod, Blackberry) โดยผ่านระบบโครงข่ายเว็บบราวเซอร์ Safari และ Internet Explorer ในการใช้งาน สำหรับอุปกรณ์ระบบปฏิบัติการ iOS จะเปรียบเสมือนกับการมอนิเตอร์หรือเป็นส่วนหนึ่งของระบบรักษาความปลอดภัยภายในครัวเรือนหรือสำนักงานด้วยระบบอินเตอร์เน็ตรีโมต

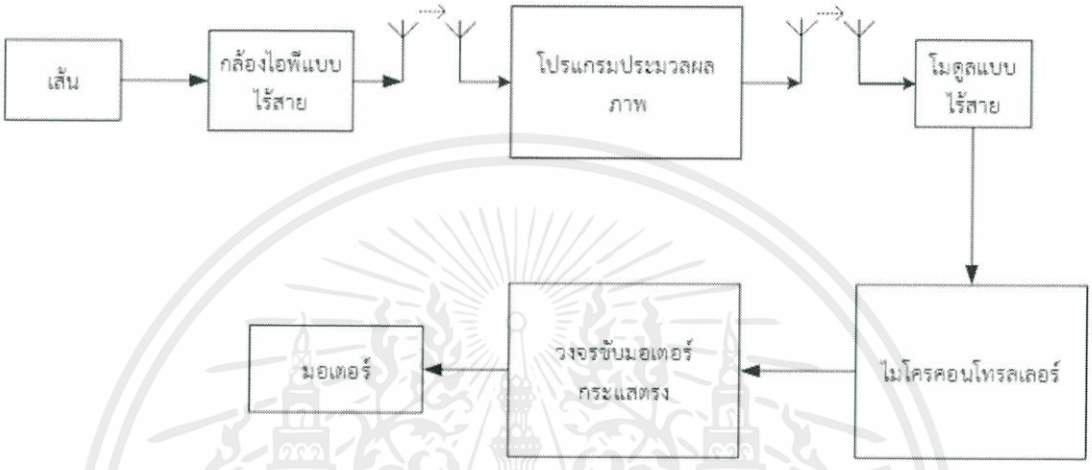
คุณสมบัติต่างๆของกล้องจัดอยู่ในตารางที่ 2.7

ตารางที่ 2.7 คุณสมบัติของ Foscam FI8909W

คุณสมบัติ	**รายละเอียด**
ความละเอียด	640 x 480 พิกเซล(300k พิกเซล)
สี	ดำ
เซ็นเซอร์ภาพ	1/4" สี CMOS เซ็นเซอร์
ความละเอียดการแสดงผล	640 x 480 พิกเซล(300k พิกเซล)
เลนส์	f: 3.6มม., F:2.4 (IR Lens)
ความเข้มแสงน้อยสุด	0.5 ลักซ์
ชนิดเลนส์	เลนส์แก้ว
มุมการมอง	60 องศา
อินพุต	ไมโครโฟนในตัว
เอาท์พุต	ลำโพงในตัว
การบันทึกเสียง	ADPCM
การบันทึกภาพ	MJPEG

เฟรมเรทภาพ	15fps(VGA),30fps(QVGA)
การกลับด้าน	แนวตั้ง / แนวนอน
ความถี่แสง	50Hz, 60Hz
วิดีโอพารามิเตอร์	สว่าง, คอนทรานซ์
อีเธอร์เน็ต	One 10/100Mbps RJ-45
มาตรฐานแบบไร้สาย	IEEE 802.11b/g
อัตราข้อมูล	802.11b: 11Mbps(Max.), 802.11g: 54Mbps(Max.)
ความปลอดภัยแบบไร้สาย	WEP&WPA Encryption
มุมเอียง	ไม่มี
อินฟราเรท	IR LEDs 5 ดวง มองเห็นในความมืดมากกว่า 7 เมตร
น้ำหนักโดยรวม	360กรัม (ขนาดกล่อง :225x172x68มม.)
น้ำหนักสุทธิ	250กรัม (รวมอุปกรณ์เสริม)
อุณหภูมิ	-10°C ~ 60°C (14°F ~ 140°F)
ความชื้น	0% ~ 90%
CPU	2.0GHZ หรือสูงกว่า
ขนาดความจำ	256MB หรือสูงกว่า
การรองรับ OS	Microsoft Windows 2000/XP/Vista / 7 / 8 / Mac
บราวเซอร์	IE 6.0, IE7.0,Firefox, Safari (ไร้เสียง), หรือบราวเซอร์มาตรฐาน

บทที่ 3
การออกแบบและการจัดทำโครงงาน



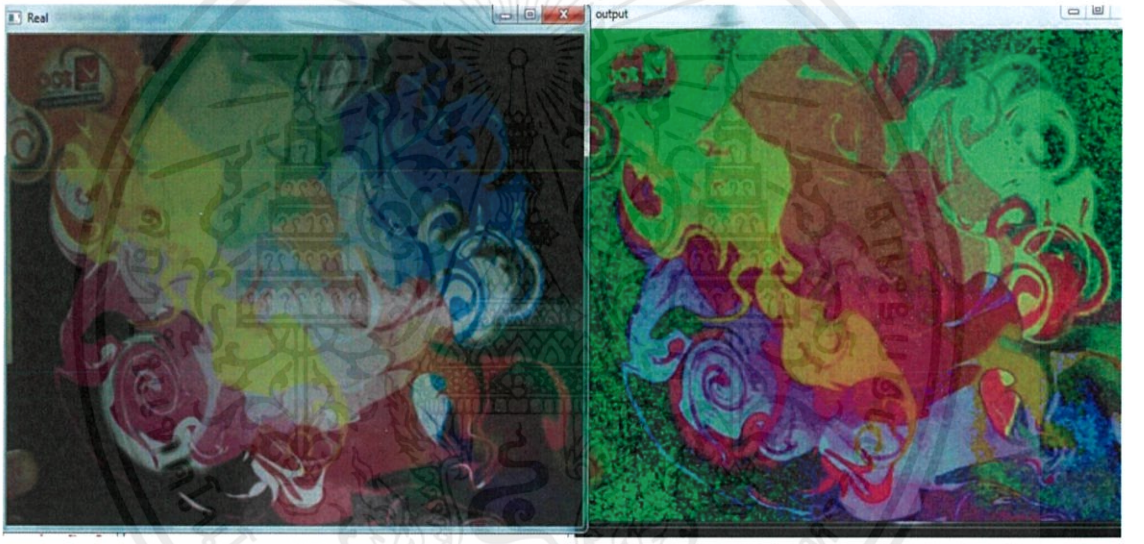
รูปที่ 3.1 บล็อกไดอะแกรมภาพรวมของโครงงาน

จากบล็อกไดอะแกรมสามารถอธิบายการทำงานของระบบได้ดังนี้ เมื่อกล้องจับภาพไปที่เส้นทาง ตำแหน่งของเส้นทางที่กล้องจับภาพได้จะมีคำสั่งซึ่งได้ทำการกำหนดไว้ คำสั่งเหล่านั้นจะถูกส่งผ่านการเชื่อมต่อแบบไร้สาย โดยที่รถจะมีโมดูลคอยรับคำสั่งที่ส่งมา จากนั้นจะลงสัญญาณคำสั่งไปที่ไมโครคอนโทรลเลอร์เพื่อเป็นคำสั่งให้กับวงจรขับมอเตอร์กระแสตรงเพื่อไปควบคุมการทำงานของมอเตอร์เพื่อควบคุมการเคลื่อนที่ของรถ

3.1 การใช้งานภาษาไพธอน ในการประมวลผลภาพ

3.1.1 การเปลี่ยนภาพ RGB เป็น HSV

ค่าสีที่อยู่ในรูปแบบของ RGB ประกอบไปด้วย ค่าสี (Color) ค่าแสง (Light) และค่าความสว่าง (Brightness) ของภาพ ซึ่งมีความยุ่งยากในการแยกแยะและค้นหาภาพสีเนื่องจากแต่ละค่าสี RGB จะมีค่าแสงและค่าความสว่างผสมอยู่ด้วย แต่แบบจำลอง HSV (Hue-Saturation-Value) จะมีการแยกความสว่างของภาพ (Luminance) ออกจากข้อมูลสี (Chromaticity) ทำให้สามารถเปรียบเทียบค่าสีได้ง่าย



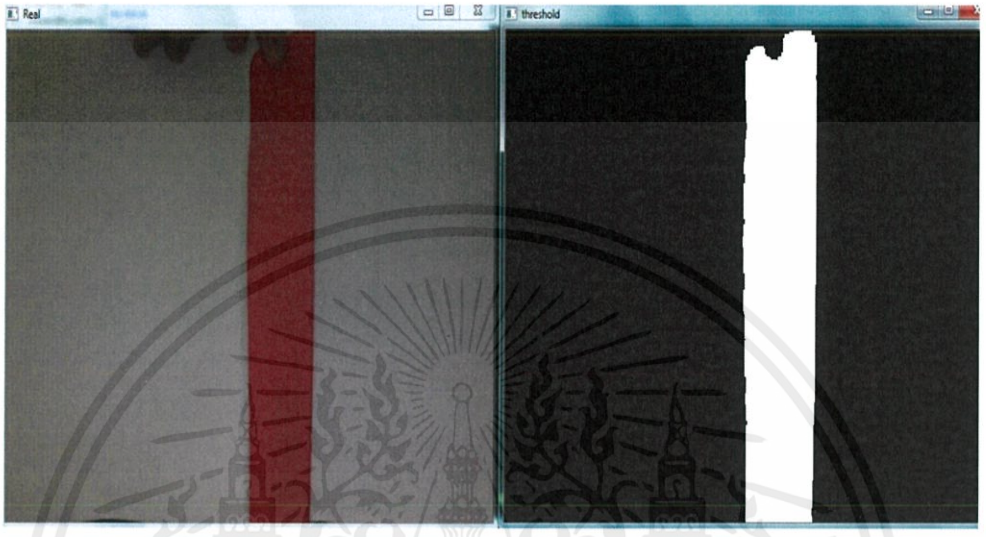
รูปที่ 3.2 การเปลี่ยนภาพจาก RGB เป็น HSV

3.1.2 กระบวนการ Segmentation

กระบวนการ Segmentation เป็นกระบวนการแยกวัตถุ หรือองค์ประกอบต่างๆ ออกจากภาพอินพุท โดยใน ซึ่งมีขั้นตอนดังนี้

1. รับภาพจากกล้อง
2. แปลงปริภูมิสีจากปริภูมิ RGB เป็นปริภูมิ HSV
3. ตั้งค่า threshold ของค่า H, S และ V ให้สอดคล้องกับสีของที่ต้องการ

4. กำหนดให้ค่าของจุดสีที่เป็นองค์ประกอบที่ต้องการพิจารณามีค่าเป็นสีขาว (255) และกำหนดให้ค่าของจุดสีที่ไม่ใช่องค์ประกอบที่ต้องการพิจารณามีค่าเป็นสีดำ (0)



รูปที่ 3.3 กระบวนการ Segmentation เพื่อหาวัตถุสีแดง

3.1.3 การหาตำแหน่งของภาพ

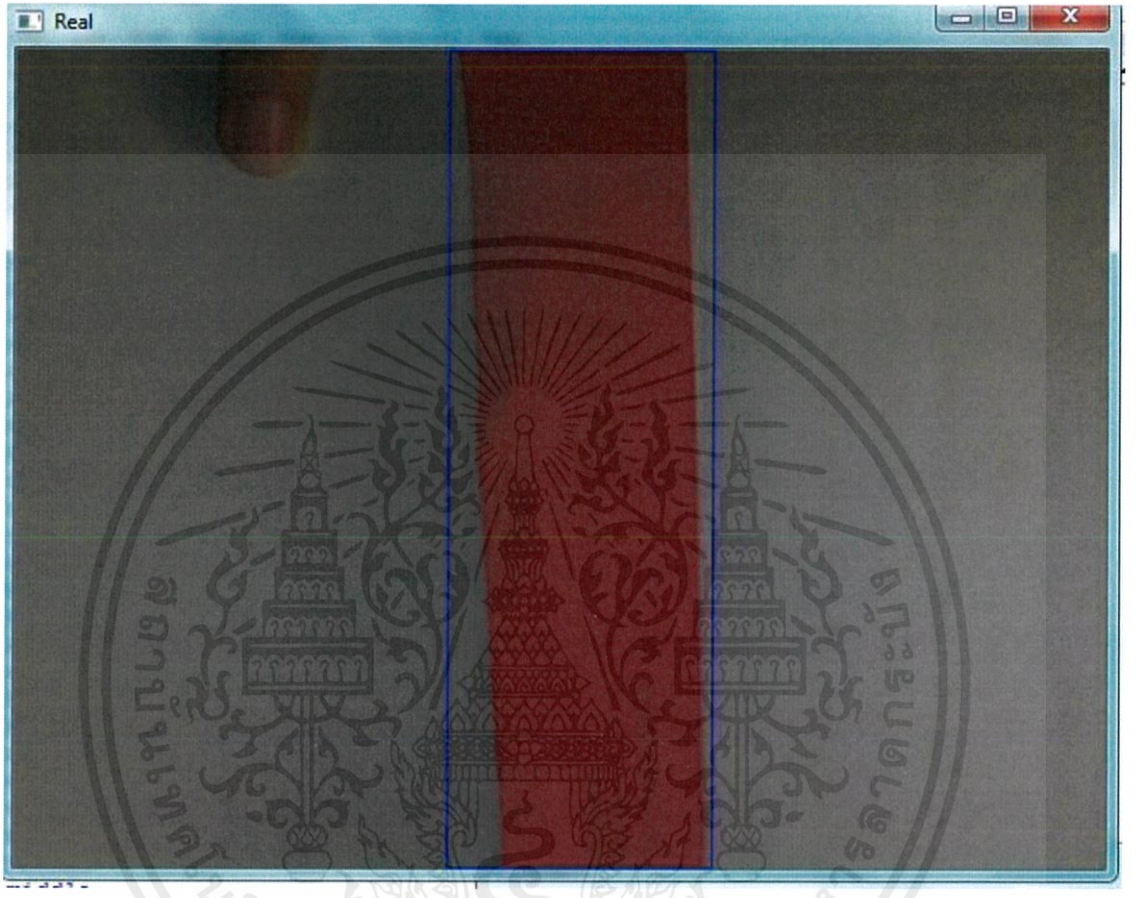
เมื่อแปลงภาพเป็นภาพขาว-ดำ (Binary Image) ซึ่งภาพขาว-ดำ นี้ จะมีสีขาวเป็นบิต 1 และสีดำเป็นบิต 0 จึงสามารถหาตำแหน่งได้จากโค้ดดังต่อไปนี้

```
def getpositions(im):

    leftmost=0
    rightmost=0
    topmost=0
    bottommost=0
    temp=0
    for i in range(im.width):
        col=cv.GetCol(im,i)
        if cv.Sum(col)[0]!=0.0:
            rightmost=i
            if temp==0:
                leftmost=i
                temp=1
    for i in range(im.height):
        row=cv.GetRow(im,i)
        if cv.Sum(row)[0]!=0.0:
            bottommost=i
            if temp==1:
                topmost=i
                temp=2
    return (leftmost,rightmost,topmost,bottommost)
```

รูปที่ 3.4 โค้ดของการหาตำแหน่งของวัตถุ

โดยจากโปรแกรมที่ได้จะได้เป็นการแสดงการหาตำแหน่งของภาพได้ เมื่อภาพของวัตถุมีสีที่กำหนดค่าไว้

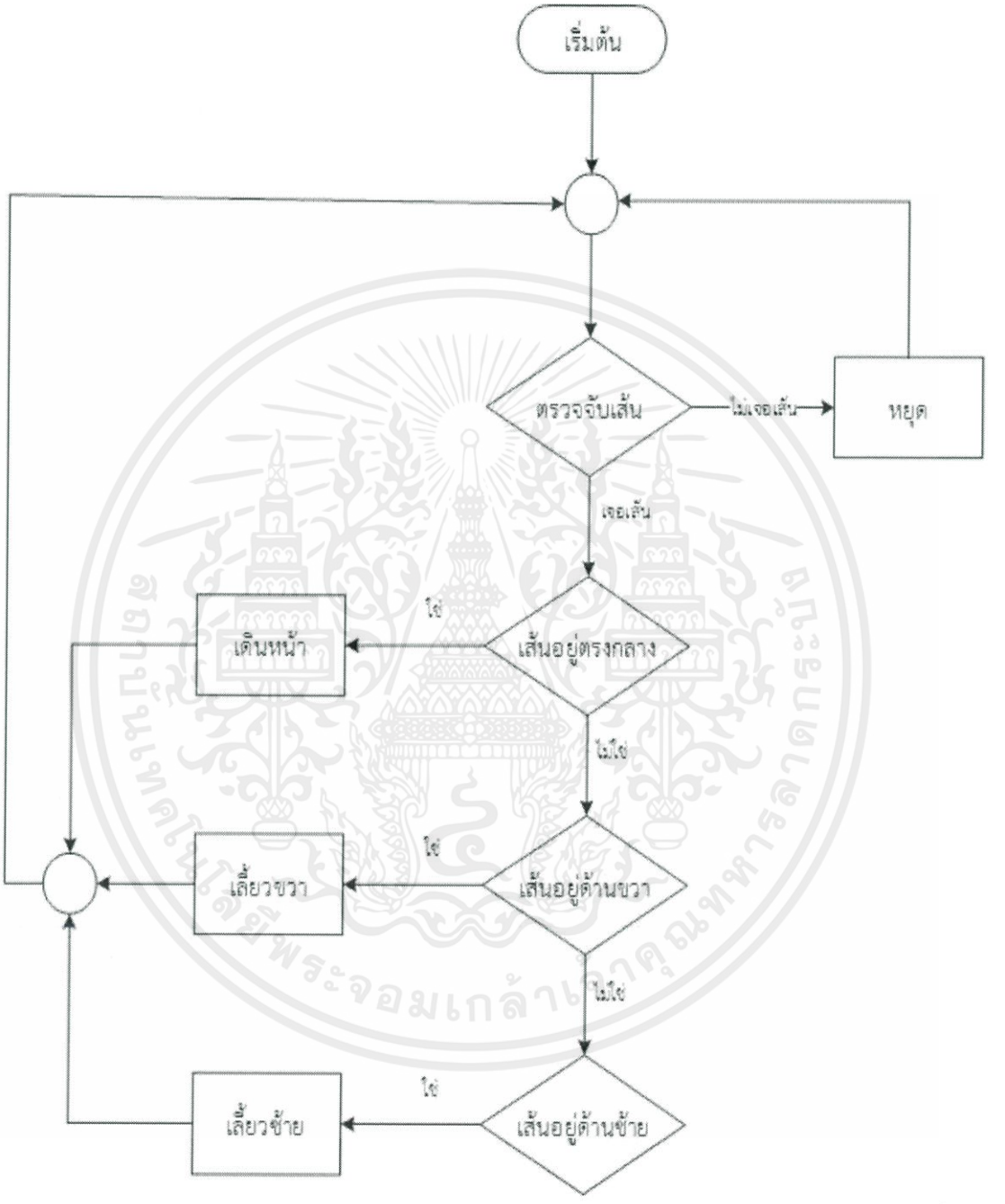


รูปที่ 3.5 การหาตำแหน่งของวัตถุที่มีสีตามที่กำหนด

3.1.4 โฟล์วชาร์ตแสดงการทำงานของโปรแกรม

การแสดงการทำงานของโปรแกรมที่ใช้ในการส่งคำสั่งควบคุมการเคลื่อนที่ของรถ โดยโปรแกรมจะจับภาพที่มีสีที่กำหนดให้เพื่อทำการส่งคำสั่งการคุมคุมไปควบคุมการเคลื่อนที่ของรถ

- โปรแกรมส่งคำสั่งการควบคุมรถ



รูปที่ 3.6 โฟล์วชาร์ตการทำงานของโปรแกรม

3.2 ส่วนของวงจรไมโครคอนโทรลเลอร์

3.2.1 การติดตั้งและการกำหนดค่าต่างๆของโมดูล

ทำการกำหนดค่าของโมดูล RN-171 โดยใช้แอพพลิเคชั่นชื่อว่า WiFlyClient ซึ่งเป็นแอพพลิเคชั่นสำหรับการใช้กำหนดค่า Configuration ต่างๆสำหรับการควบคุมการทำงานของโมดูล RN-171 ที่จะใช้โดยทำการเชื่อมต่อกับโมดูลที่จะใช้เป็นตัวรับคำสั่ง

ขั้นตอนการกำหนดค่าต่างๆ ของโมดูล RN-171

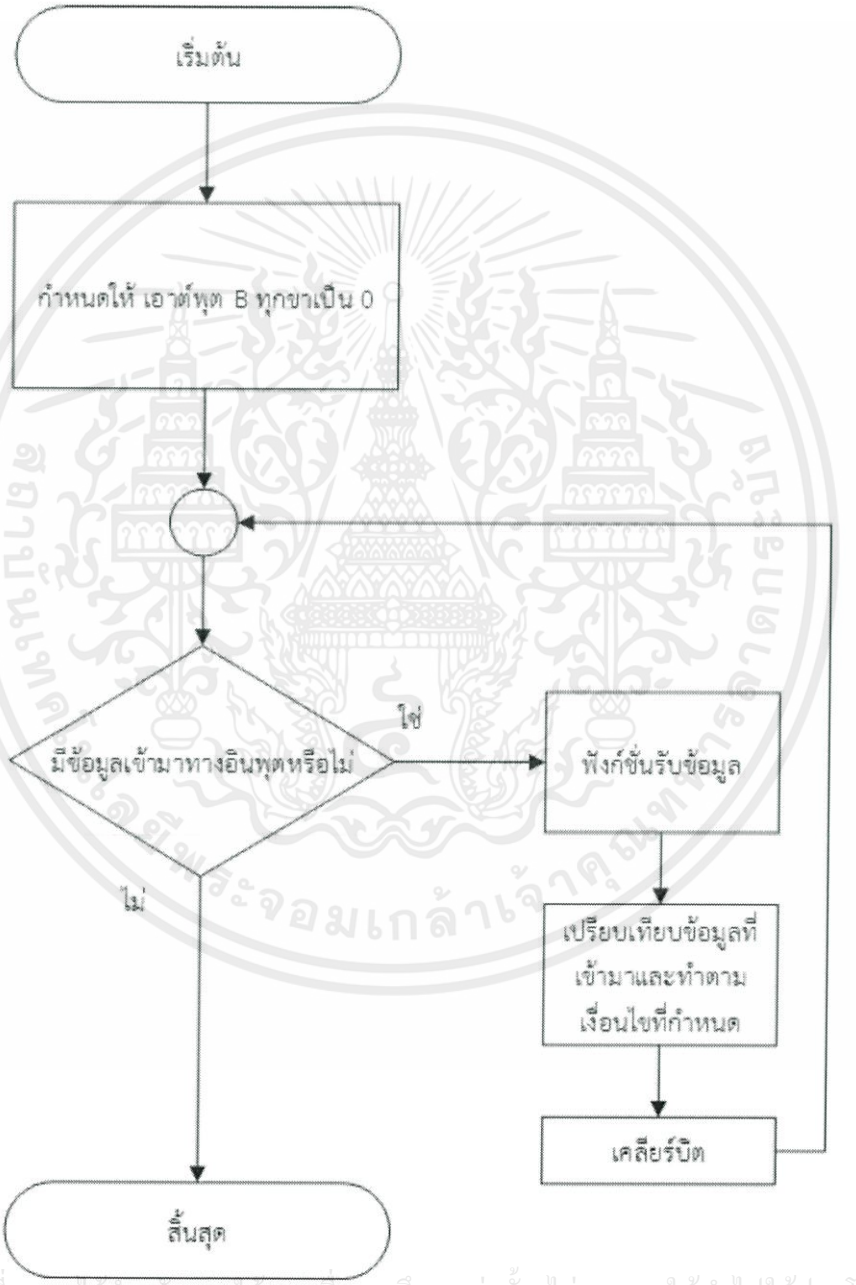
1. ทำการเชื่อมต่อโมดูลกับแอพพลิเคชั่น WiFlyCient
2. สังเกตหลอดไฟ LED จะเห็นหลอดไฟแสดงสถานการณ์เชื่อมต่อระหว่างโมดูล RN-171 กับภาคส่งที่กำหนดคำสั่งด้วยแอพพลิเคชั่น
3. ทำการกำหนดค่าต่างๆ โดยกำหนด Protocol และ Port Number ของตัวโมดูลเพื่อเชื่อมกับภาคส่งโดยโมดูลตั้งค่าดังนี้

```
set ip proto 1              //ให้ UDP เป็น Protocol
set ip host 169.254.11.1    //ตั้งค่า IP address ของโมดูล
set ip local 5005           //ตั้งค่า Local
save                        //บันทึกค่า
reboot                      //reboot โมดูลเพื่อให้แสดงค่าตามที่ตั้งไว้
```

4. ค่าที่เรากำหนดให้กับตัวโมดูลนั้นจะถูกเก็บไว้ในตังเครื่องอย่างถาวร จนกว่าจะมีการรีเซ็ตค่าของโมดูลใหม่ถึงแม้ว่าไม่มีการจ่ายไฟให้กับตัวโมดูลก็ตาม หากต้องการเปลี่ยนแปลงค่าใหม่ต้องทำการรีเซ็ตค่าแล้วทำตามขั้นตอนตามข้างต้น

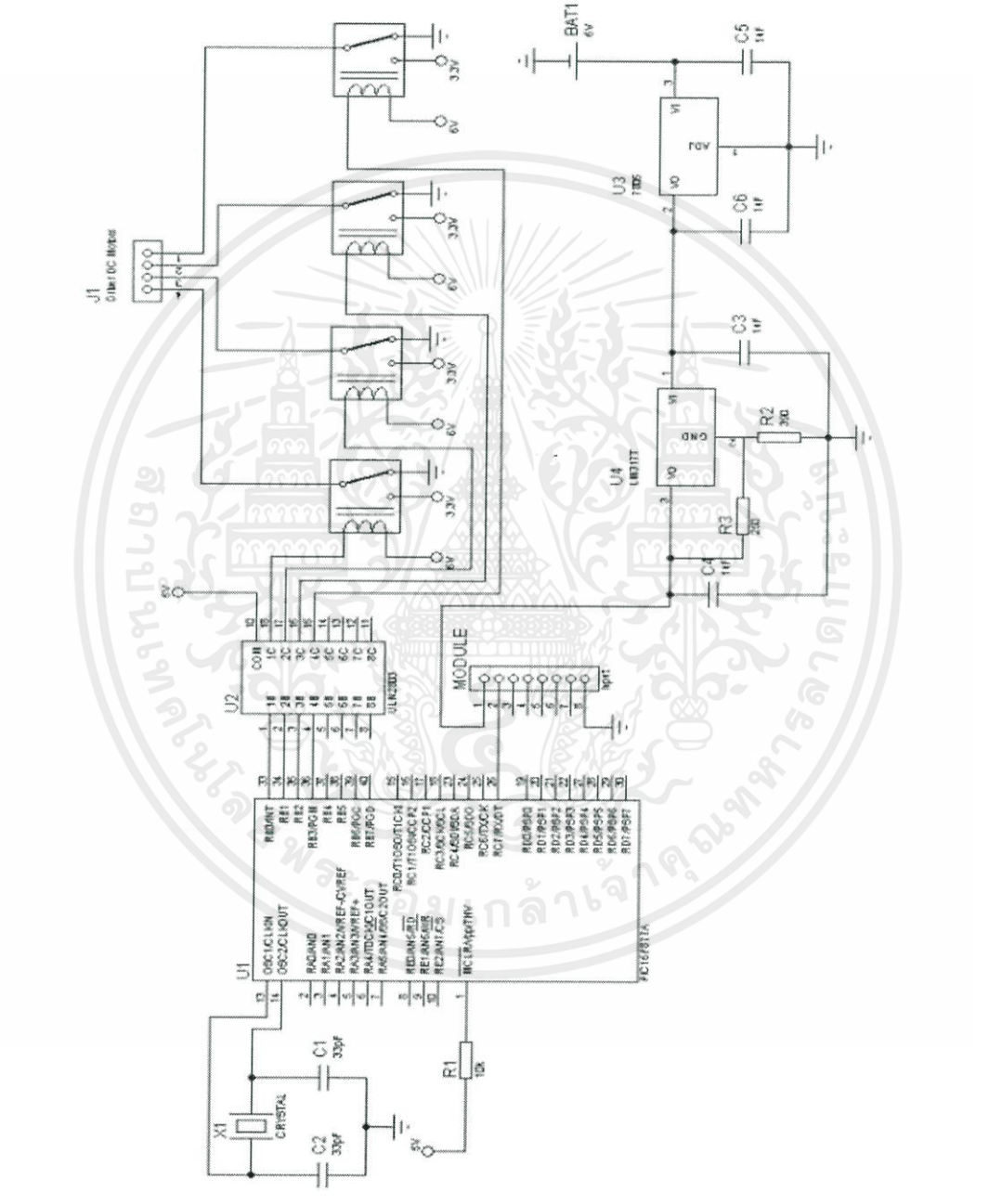
3.2.2 การทำงานของการรับคำสั่งในการควบคุม

ไมโครคอนโทรลเลอร์จากจะทำหน้าที่รับคำสั่งที่ได้รับจากโมดูลมาประมวลเป็นคำสั่งต่อไป โดยจะเป็นตัวกำหนดแรงดันในการเคลื่อนที่ของมอเตอร์กระแสตรง เพื่อควบคุมการเคลื่อนที่ของรถ



รูปที่ 3.7 โฟล์วชาร์ตแสดงการทำงานของไมโครคอนโทรลเลอร์ในการควบคุมการเคลื่อนที่ของรถ

จากรูปที่ 3.6 แสดงการทำงานของวงจรรวมในการควบคุมการเคลื่อนที่ของรถ และในรูปที่ 3.7 จะแสดงถึงวงจรภาพรวมของวงจรไมโครคอนโทรลเลอร์ที่ควบคุมการเคลื่อนที่ของรถ



รูปที่ 3.8 วงจรรวมของวงจรควบคุมการเคลื่อนที่ของรถ

3.3 เครื่องมือที่ใช้ในการทดลอง

1. คอมพิวเตอร์ (Window OS)
2. ออสซิลโลสโคป
3. โมดูล WiFly RN-171
4. แบตเตอรี่แห้ง 6V 4.5Ah
5. รถบังคับมอเตอร์กระแสตรง
6. ถ่านอัลคาไลน์
7. กระดาษกาวสีแดง สีเหลือง และ สีน้ำเงิน
8. IP Camera

3.4 การจัดเก็บผลการทดลอง

3.4.1 การทดสอบภาพจากกล้องและสัญญาณข้อมูลที่ออกมาจากการประมวลผล

1. ภาพและสัญญาณคำสั่งให้เคลื่อนที่ไปข้างหน้า
2. ภาพและสัญญาณคำสั่งให้เคลื่อนที่เลี้ยวซ้าย
3. ภาพและสัญญาณคำสั่งให้เคลื่อนที่เลี้ยวขวา

3.4.2 การทดสอบสัญญาณข้อมูลที่รับโดยใช้ออสซิลโลสโคป

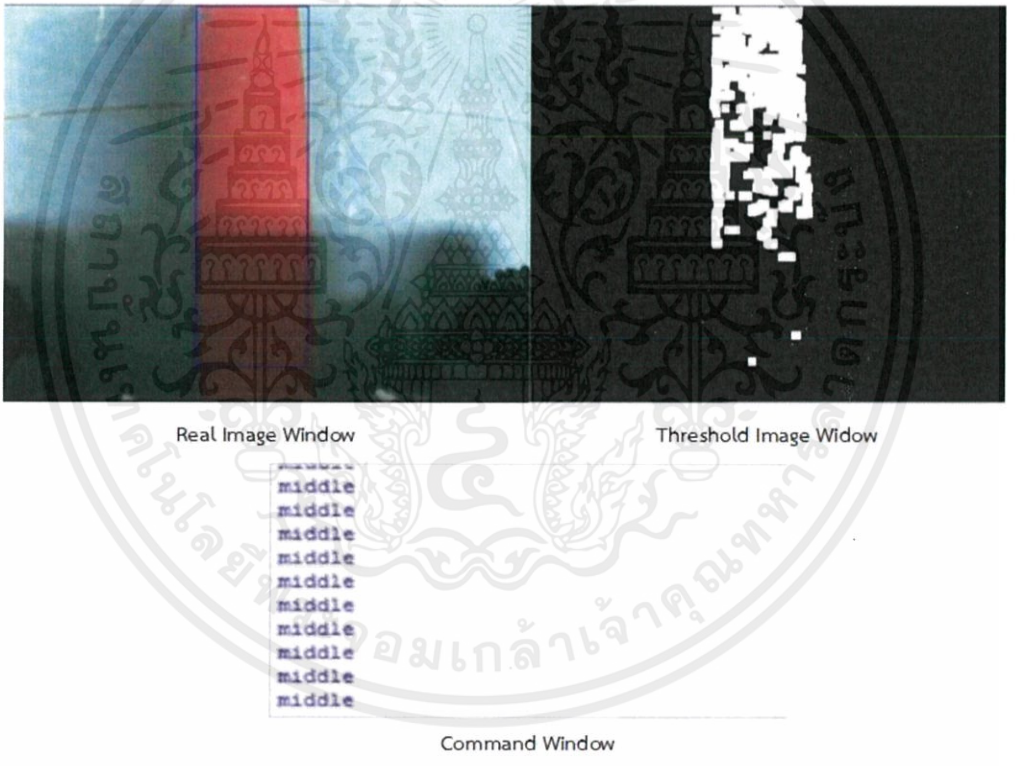
1. สัญญาณคำสั่งเดินหน้า
2. สัญญาณคำสั่งเลี้ยวซ้าย
3. สัญญาณคำสั่งเลี้ยวขวา

บทที่ 4

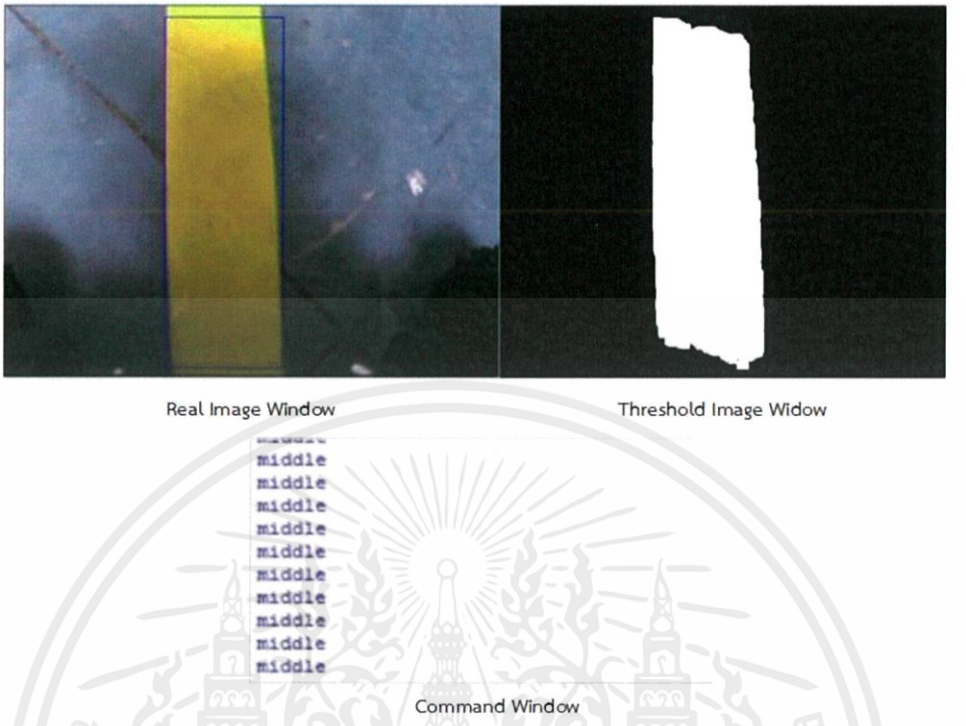
ผลการทดลอง

4.1 ทดสอบการส่งข้อมูลคำสั่ง

ทดลองการประมวลผลสัญญาณภาพที่ใช้กับโครงงาน โดยใช้โปรแกรมภาษา python ที่ใช้ในการประมวลผลสัญญาณภาพ โดยใช้ภาพจำลองเส้นทางในการทดลองโปรแกรม



รูปที่ 4.1 เส้นทางสีแดงที่ใช้ในการประมวลผลภาพเมื่ออยู่ตรงกลาง



รูปที่ 4.2 เส้นทางสีเหลืองที่ใช้ในการประมวลผลภาพเมื่ออยู่ตรงกลาง

รูปที่ 4.3 เส้นทางสีน้ำเงินที่ใช้ในการประมวลผลภาพเมื่ออยู่ตรงกลาง

จากรูปที่ 4.1, 4.2 และ 4.3 รูปเส้นทางสีต่างๆ ที่ใช้ในการประมวลผลภาพเมื่อเส้นอยู่ตรงกลาง ซึ่งจะเป็นคำสั่งในรถเคลื่อนที่ไปข้างหน้า

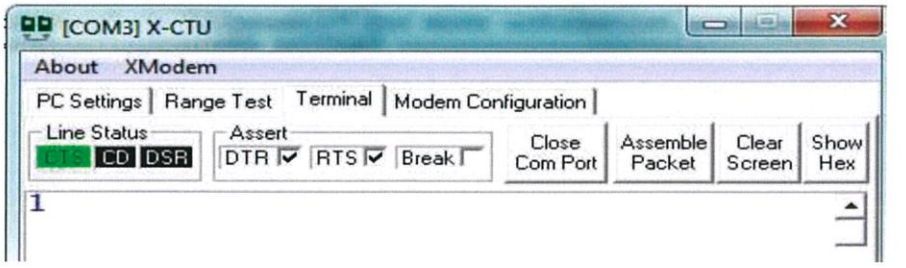
```
import Image
import urllib2
im = Image.open("c:\\test.bmp")
def gen(im):
        for x in range(im.size[0]):
                px = im.getpixel((x, 0))
                if px[0] > 200 and px[1] < 40 and px[2] < 40 :
                    for n in range(x, im.size[0]):
                        px = im.getpixel((n, 0))
                        if px[0] > 0 and px[1] > 0 and px[2] > 0:
                            if x<116 and n<281 :
                                UDP_IP = "192.168.1.135"
                                UDP_PORT = 2000
                                sock = socket.socket(socket.AF_INET,
                                sock.sendto("2", (UDP_IP, UDP_PORT))
                                return(2)
                            elif x>136 and n>291 :
                                UDP_IP = "192.168.1.135"
                                UDP_PORT = 2000
                                sock = socket.socket(socket.AF_INET,
                                sock.sendto("3", (UDP_IP, UDP_PORT))
                                return(3)
                            else :
                                UDP_IP = "192.168.1.135"
                                UDP_PORT = 2000
                                sock = socket.socket(socket.AF_INET,
                                sock.sendto("1", (UDP_IP, UDP_PORT))
                                return(1)

m = gen(im)
print m
```

```
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
1
>>>
```

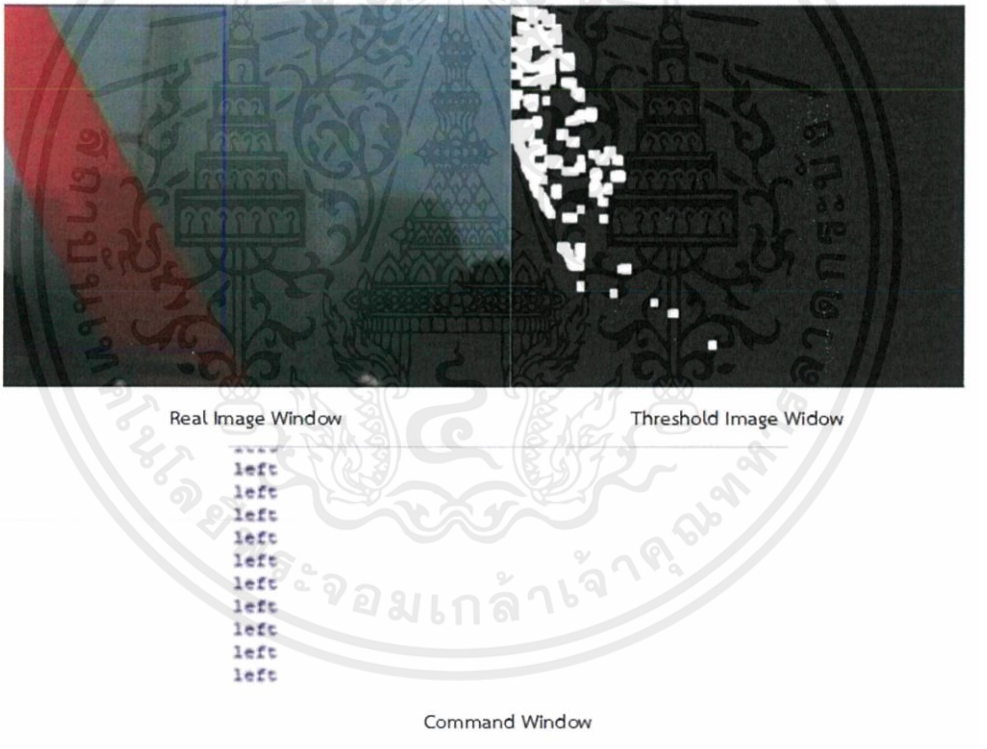
รูปที่ 4.4 ส่วนของโปรแกรมเมื่อเส้นอยู่ตรงกลาง

จากรูปที่ 4.4 เป็นผลของโปรแกรมเมื่อเส้นอยู่ตรงกลางของกล้อง จะส่งเป็นเลข 1 (ฐานสิบ) เข้าสู่โมดูลแล้วส่งคำสั่งไปให้ไมโครคอนโทรลเลอร์ต่อไป เพื่อบังคับวงจรขับมอเตอร์กระแสตรง

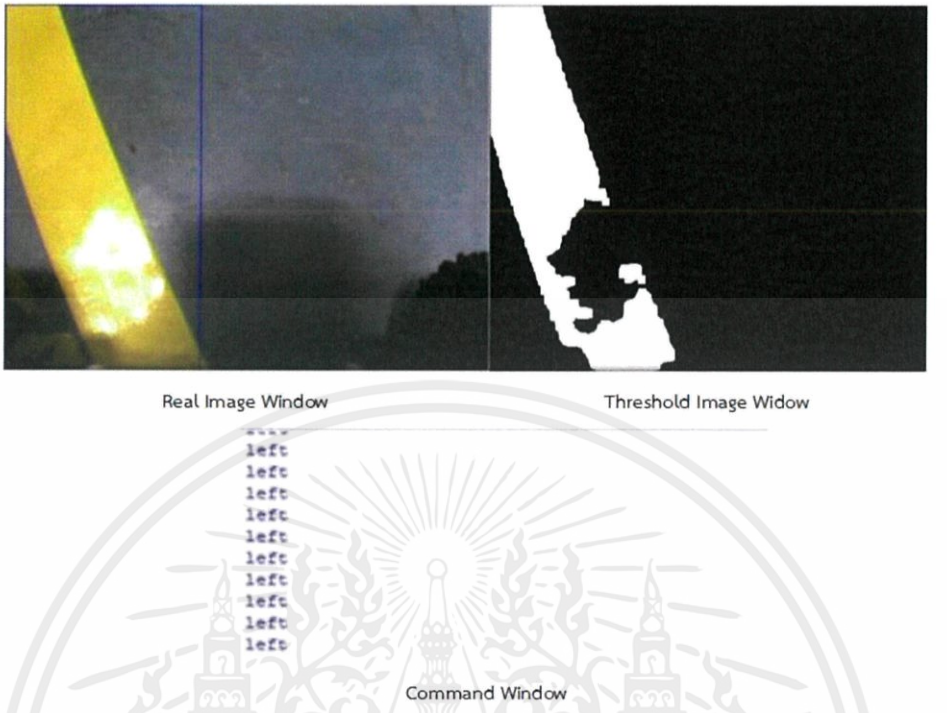


รูปที่ 4.5 ผลที่ส่งเข้าโมดูลเมื่อเส้นอยู่ตรงกลาง

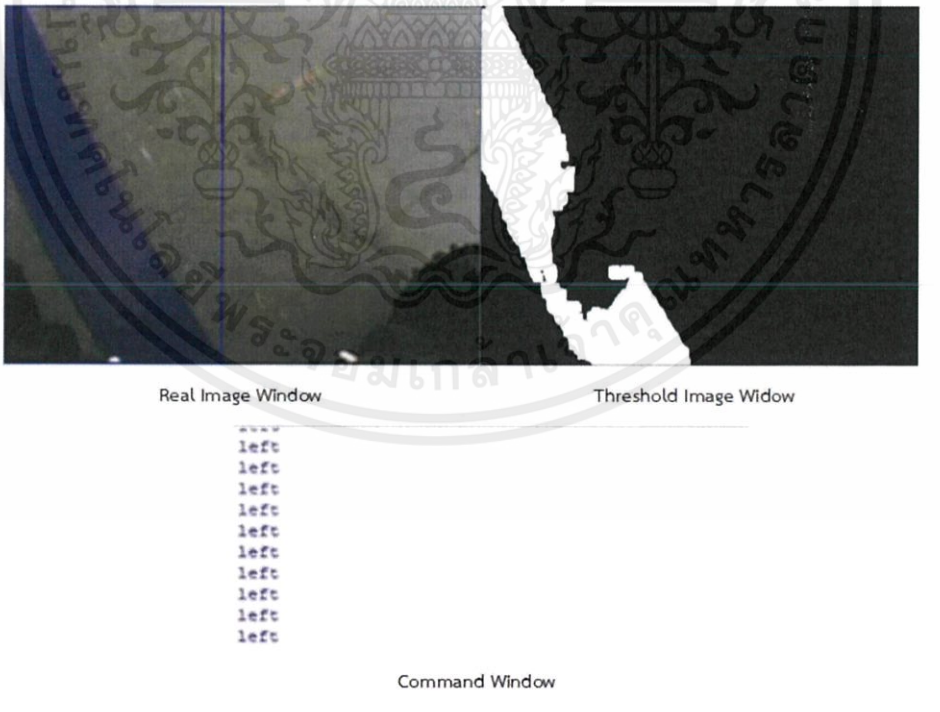
จากรูปที่ 4.5 เป็นผลที่ถูกส่งมาจากโปรแกรมประมวลผลภาพเข้าสู่โมดูล เมื่อเส้นอยู่ตรงกลางของกล้อง โดยจะส่งเป็นเลข 1 (ฐานสิบ)



รูปที่ 4.6 เส้นทางสีแดงที่ใช้ในการประมวลผลภาพเมื่ออยู่ด้านซ้าย

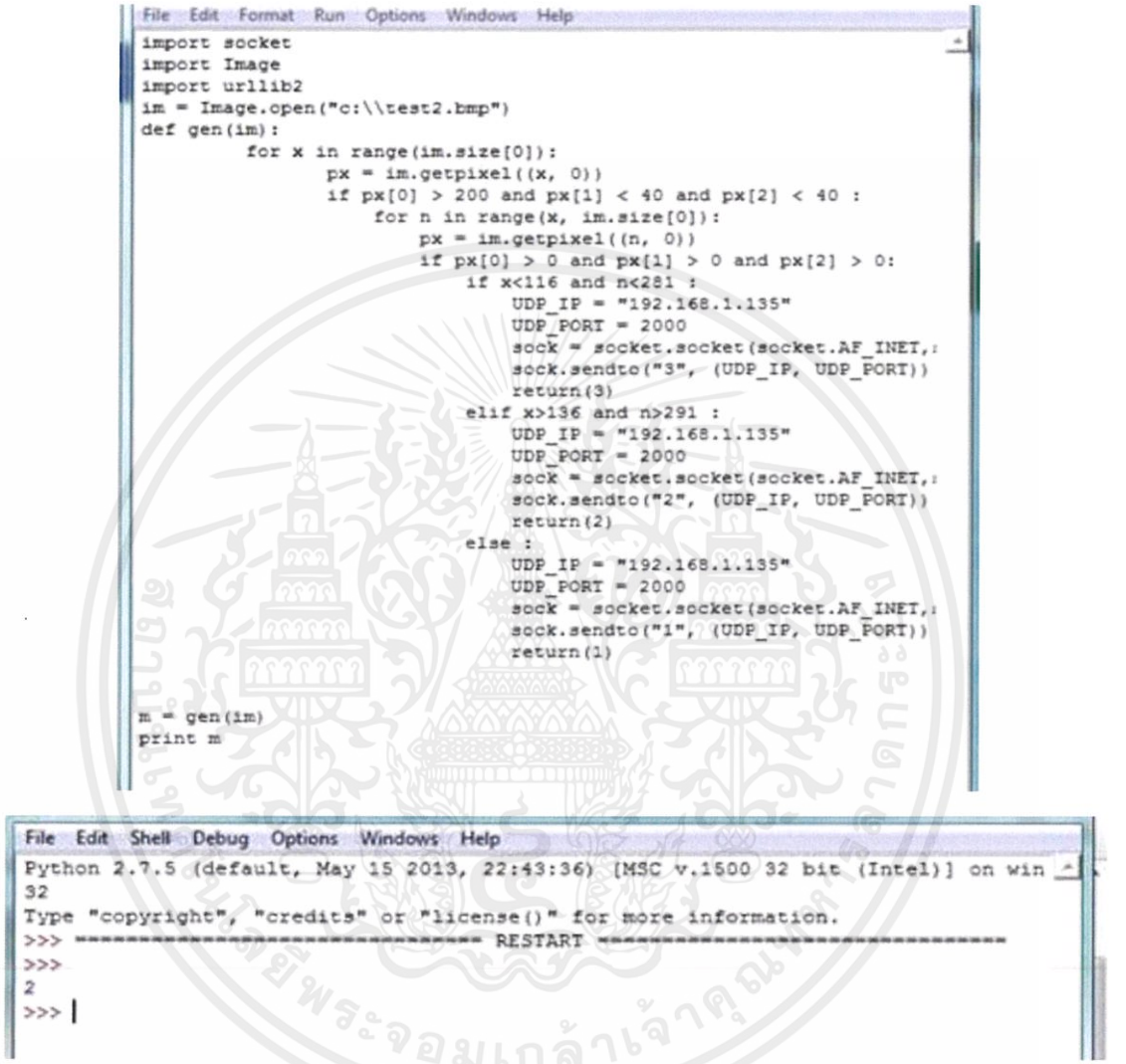


รูปที่ 4.7 เส้นทางสีเหลืองที่ใช้ในการประมวลผลภาพเมื่ออยู่ด้านซ้าย



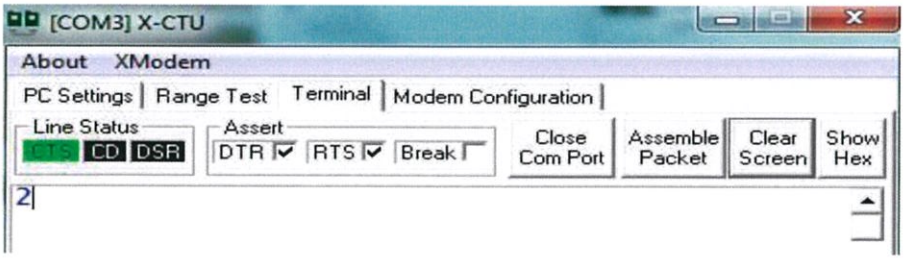
รูปที่ 4.8 เส้นทางสีน้ำเงินที่ใช้ในการประมวลผลภาพเมื่ออยู่ด้านซ้าย

จากรูปที่ 4.6, 4.7 และ 4.8 รูปเส้นทางสีต่างๆ ที่ใช้ในการประมวลผลภาพเมื่อเส้นอยู่ด้านซ้าย ซึ่งจะเป็นคำสั่งในรถเคลื่อนที่เลี้ยวซ้าย



รูปที่ 4.9 ส่วนของโปรแกรมเมื่อเส้นอยู่ด้านซ้าย

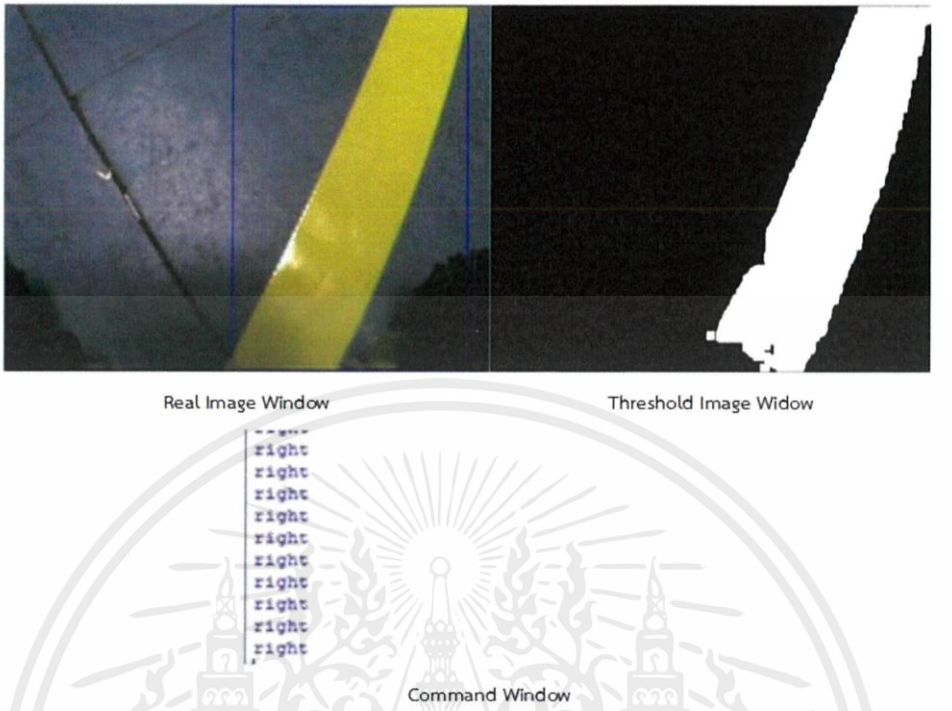
จากรูปที่ 4.9 เป็นผลของโปรแกรมเมื่อเส้นอยู่ด้านซ้ายของกล้อง จะส่งผลเป็นเลข 2 (ฐาน 10) เข้าสู่โมดูลจึงส่งไปไมโครคอนโทรลเลอร์ต่อไป เพื่อบังคับวงจรขับมอเตอร์กระแสตรงให้รถเลี้ยวซ้าย



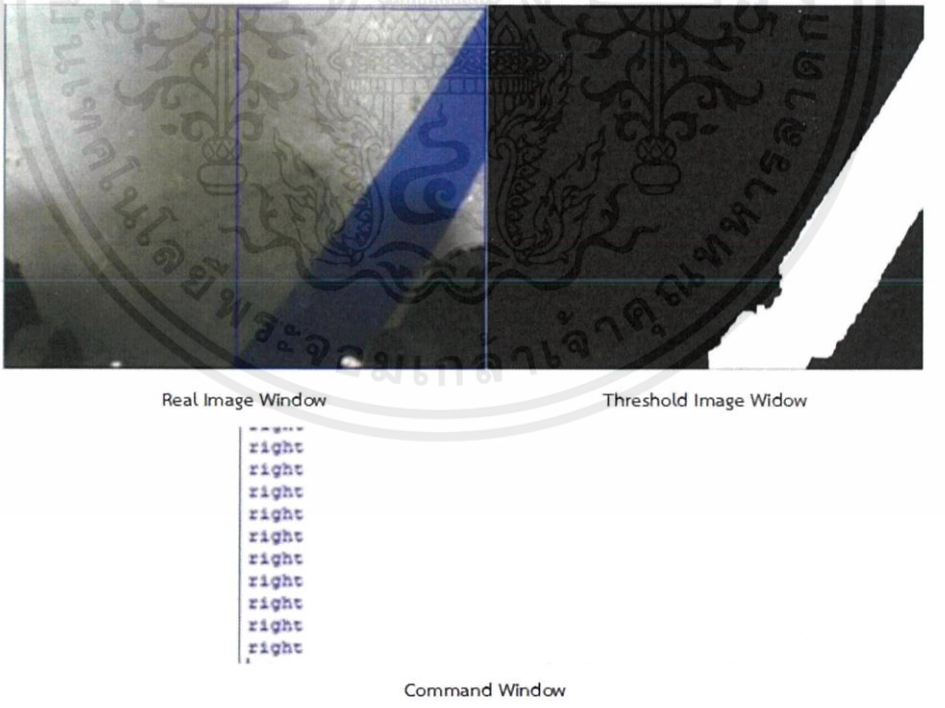
รูปที่ 4.10 ผลที่ส่งเข้าโมดูลเมื่อเส้นอยู่ด้านซ้าย

จากรูปที่ 4.10 เป็นผลที่ถูกส่งมาจากโปรแกรมประมวลผลภาพเข้าสู่โมดูล เมื่อเส้นอยู่ด้านซ้ายของกล้อง โดยจะส่งเป็นเลข 2 (ฐานสิบ)

รูปที่ 4.11 เส้นสีแดงที่ใช้ในการประมวลผลภาพเมื่อเส้นอยู่ด้านขวา

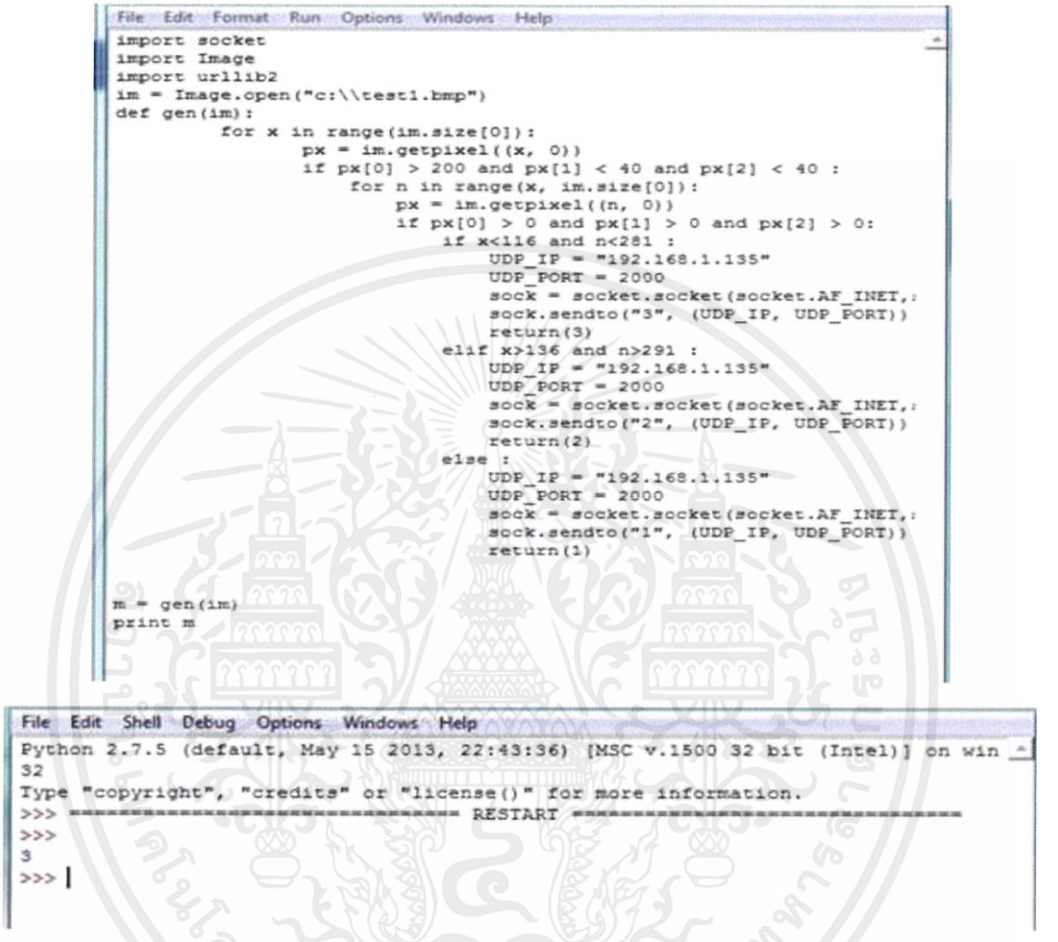


รูปที่ 4.12 เส้นสีเหลืองที่ใช้ในการประมวลผลภาพเมื่อเส้นอยู่ด้านขวา



รูปที่ 4.13 เส้นสีน้ำเงินที่ใช้ในการประมวลผลภาพเมื่อเส้นอยู่ด้านขวา

จากรูปที่ 4.11, 4.12 และ 4.13 เป็นรูปเส้นสีต่างๆ ที่ใช้ในการประมวลผลภาพเมื่อเส้นอยู่ด้านขวา ซึ่งจะเป็นผลให้โปรแกรมสั่งให้รถเลี้ยวขวา



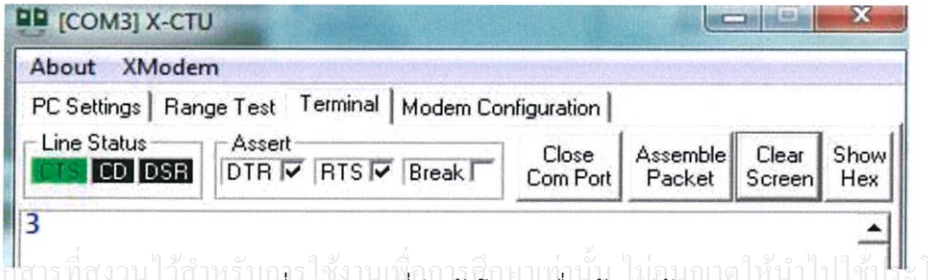
```
File  Edit  Format  Run  Options  Windows  Help
import socket
import Image
import urllib2
im = Image.open("c:\\test1.bmp")
def gen(im):
        for x in range(im.size[0]):
                px = im.getpixel((x, 0))
                if px[0] > 200 and px[1] < 40 and px[2] < 40 :
                    for n in range(x, im.size[0]):
                        px = im.getpixel((n, 0))
                        if px[0] > 0 and px[1] > 0 and px[2] > 0:
                            if x<116 and n<281 :
                                UDP_IP = "192.168.1.135"
                                UDP_PORT = 2000
                                sock = socket.socket(socket.AF_INET,
                                sock.sendto("3", (UDP_IP, UDP_PORT))
                                return(3)
                            elif x>136 and n>291 :
                                UDP_IP = "192.168.1.135"
                                UDP_PORT = 2000
                                sock = socket.socket(socket.AF_INET,
                                sock.sendto("2", (UDP_IP, UDP_PORT))
                                return(2)
                            else :
                                UDP_IP = "192.168.1.135"
                                UDP_PORT = 2000
                                sock = socket.socket(socket.AF_INET,
                                sock.sendto("1", (UDP_IP, UDP_PORT))
                                return(1)

m = gen(im)
print m
```

```
File  Edit  Shell  Debug  Options  Windows  Help
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
3
>>>
```

รูปที่ 4.14 ส่วนของโปรแกรมเมื่อเส้นอยู่ด้านขวา

จากรูปที่ 4.14 เป็นผลที่ถูกส่งมาจากโปรแกรมประมวลผลภาพเข้าสู่โมดูล เมื่อเส้นอยู่ด้านขวาของกล้อง โดยจะส่งเป็นเลข 3 (ฐานสิบ)

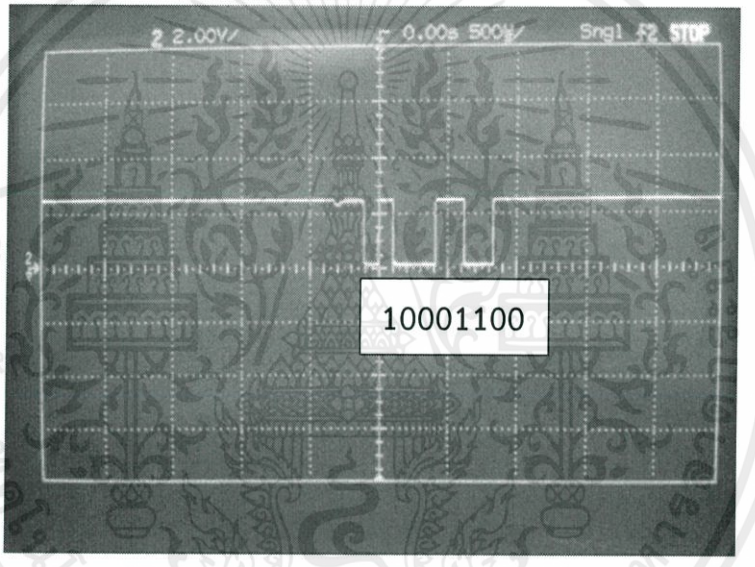


รูปที่ 4.15 ผลที่ส่งเข้าโมดูลเมื่อเส้นอยู่ด้านขวา

จากรูปที่ 4.15 เป็นผลที่ถูกส่งมาจากโปรแกรมประมวลผลภาพเข้าสู่โมดูล เมื่อเส้นอยู่ด้านขวาของกล้อง โดยจะส่งเป็นเลข 3 (ฐานสิบ)

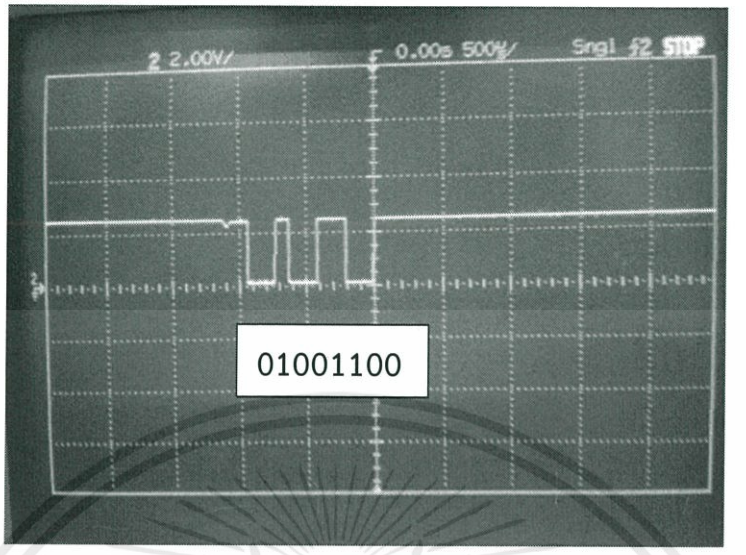
4.2 การทดสอบการรับสัญญาณข้อมูลคำสั่ง

ทำการทดลองโดยนำภาพการจำลองเส้นทางเพื่อส่งคำสั่งควบคุมต่างๆไปยังไมโครคอนโทรลเลอร์ซึ่งเป็นข้อมูลแบบ 8 บิต ที่ส่งออกจากโมดูลจะส่งข้อมูลเป็นแบบอนุกรม สามารถวัดสัญญาณที่ออกมาได้ดังนี้



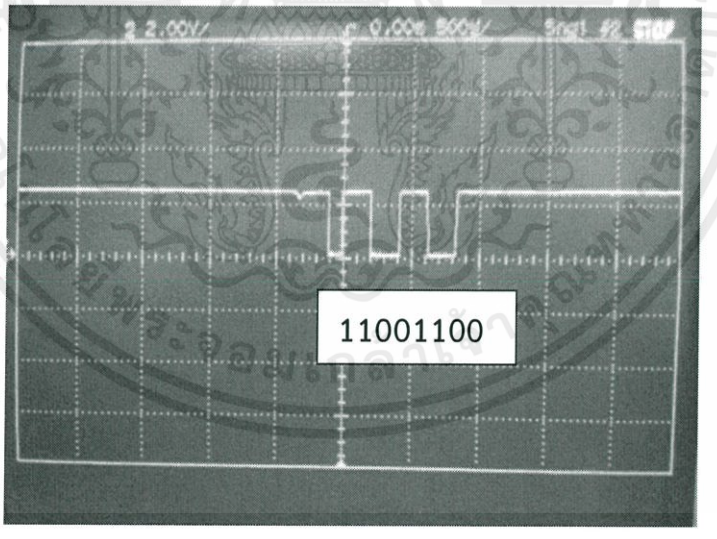
รูปที่ 4.16 สัญญาณข้อมูลคำสั่งเมื่อสั่งให้รถเดินหน้า

จากรูปที่ 4.16 สัญญาณข้อมูลที่รับมาจากโปรแกรมจะส่งอยู่ในรูปของเลขฐานสอง โดยสัญญาณคำสั่งให้เดินหน้าส่งมาเป็น 10001100



รูปที่ 4.17 สัญญาณข้อมูลคำสั่งเมื่อสั่งให้รถเลี้ยวซ้าย

จากรูปที่ 4.17 สัญญาณข้อมูลที่รับมาจากโปรแกรมจะส่งอยู่ในรูปของเลขฐานสอง โดยสัญญาณคำสั่งให้เลี้ยวซ้ายส่งมาเป็น 01001100



รูปที่ 4.18 สัญญาณข้อมูลคำสั่งเมื่อสั่งให้รถเลี้ยวขวา

จากรูปที่ 4.18 สัญญาณข้อมูลที่รับมาจากโปรแกรมจะส่งอยู่ในรูปของเลขฐานสอง โดยสัญญาณคำสั่งให้เลี้ยวขวาส่งมาเป็น 11001100

*หมายเหตุ สามารถเข้าชมผลการทดลองได้ที่ :https://www.youtube.com/watch?v=YTRlz5L4jJA

บทที่ 5

สรุปผลและข้อเสนอแนะ

5.1 สรุปผล

จากการทดลองการควบคุมการเคลื่อนที่ของรถโดยใช้การประมวลผลสัญญาณภาพ โดยส่งคำสั่งที่ควบคุมไปให้ไมโครคอนโทรลเลอร์เพื่อควบคุมทิศทางจากการประมวลผลสัญญาณภาพสามารถขับเคลื่อนการเคลื่อนที่ของรถเป็นไปตามที่กำหนดตามเส้นทางได้

1. สามารถเขียนโปรแกรมไมโครคอนโทรลเลอร์โดยใช้ภาษา C ในการควบคุมการสั่งการทิศทางของรถ

2. สามารถใช้อุปกรณ์อิเล็กทรอนิกส์ในการนำมาสร้างเป็นวงจรรับคำสั่งจากไมโครคอนโทรลเลอร์

3. ใช้การรับคำสั่งในการควบคุมคุมรถโดยผ่านโมดูล RN-171 ซึ่งเป็นการรับสัญญาณข้อมูลแบบไร้สายจากการรับคำสั่งจากการประมวลผลสัญญาณภาพ

4. สามารถเขียนโปรแกรมคำสั่งโดยใช้ภาษา python ในการใช้การประมวลผลสัญญาณภาพ

5.2 ข้อเสนอแนะ

1. ในการรับคำสั่งข้อมูลควรเป็นกล้องไร้สายที่ติดกับตัวรถ

2. ในการทดลองการวิ่งของรถควรหาพื้นที่ที่รถสามารถเข้าไปถึงได้

3. ในการทดลองเขียนโปรแกรมรถมีการเคลื่อนที่ที่หลากหลายกว่าที่เป็นอยู่ เช่น มีการประมวลผลสัญญาณภาพแบบต่างๆ แล้วสามารถใช้คำสั่งนั้นในการพัฒนาใช้ประโยชน์ได้ในชีวิตจริง

4. การพัฒนาการใช้งานของรถวิ่งตามเส้นโดยใช้การประมวลผลสัญญาณภาพสามารถนำไปพัฒนาเป็นการขนส่ง เช่น ขนส่งวัตถุดิบหรือผลิตภัณฑ์ เป็นต้น

5. การใช้เส้นสีดำหรือสีขาวทำให้เกิดปัญหาในการประมวลผลภาพที่ได้จากกล้อง โดยสำขาวและสีดำเป็นสีพื้นของสิ่งต่างๆ อย่างเช่นสีดำ หากพบเงา โปรแกรมจะตรวจจับได้แล้วนำมาวิเคราะห์ทำให้เกิดการวิเคราะห์ที่ไม่แม่นยำ หรือสีขาวที่เกิดจากการสะท้อนของหลอดไฟ เป็นต้น

บรรณานุกรม

[1] ทิพย์กลม ฉ่ำบุญรอด,ธัญรัศม์ สัชฌุกร,นพรัตน์ ใจวงศ์ศา. “ระบบควบคุมรถบังคับวิทยุด้วยคอมพิวเตอร์ระยะไกลในย่านความถี่ 2.4 GHz”. ปริญญานิพนธ์วิศวกรรมศาสตร์, สาขาวิชาวิศวกรรมโทรคมนาคม คณะวิศวกรรมศาสตร์, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2554.

[2]ดอนสัน ปงผาบ,ทิพวัลย์ คำน้ำนอง. *ไมโครคอนโทรลเลอร์ PIC และการประยุกต์การใช้งาน.* พิมพ์ครั้งที่ 4. กรุงเทพฯ : สมาคมส่งเสริมเทคโนโลยี(ไทย-ญี่ปุ่น), 2550.

[3]Roving Networks Inc. “RN-171.” http://www.rovingnetworks.com/products/RN_171.

[4]DATASHEETARCHIVE. “LM314.” http://www.datasheetarchive.com/lm314-datasheet.html.

[5]มหาวิทยาลัยสงขลานครินทร์. “ระบบสี RGB HSV.” http://fivedots.coe.psu.ac.th/~montri/Teaching/image/chap1.htm.

[6]ISLT. “ระบบสี RGB HSV.” http://obor.us/research-file/senior/2553/hand/final.docx.

[7] Python Software Foundation. “basic python for beginners.” http://www.python.org/about/gettingstarted/

[8] จักรพงษ์ อึ้งอาพร,และทศพร คาเนตร. “หุ่นยนต์วิ่งตามเส้น.” วิทยานิพนธ์ปริญญาวิศวกรรมศาสตรบัณฑิต, ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์, มหาวิทยาลัยขอนแก่น, 2553.

[9] Foscam Digital Technologies LLC - A United States Limited Liability Corporation. “IP Camera Foscam FI8909W.” http://foscam.us/foscam-fi8909w-na-slim-mini-wireless-ip-camera-24.html.

ภาคผนวก

PIC16F87X
Data Sheet

28/40-Pin 8-Bit CMOS FLASH Microcontrollers

Trademarks

DNV Certification, Inc. USA

ANSI-RAB QMS ACCREDITED

DNV

DNV MSC The Netherlands Accredited by the RvA

ISO 9001 / QS-9000 REGISTERED FIRM

Microchip received QS-9000 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona in July 1999. The Company's quality system processes and procedures are QS-9000 compliant for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs and microperipheral products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001 certified.

MICROCHIP

PIC16F87X

28/40-Pin 8-Bit CMOS FLASH Microcontrollers

Devices Included in this Data Sheet:

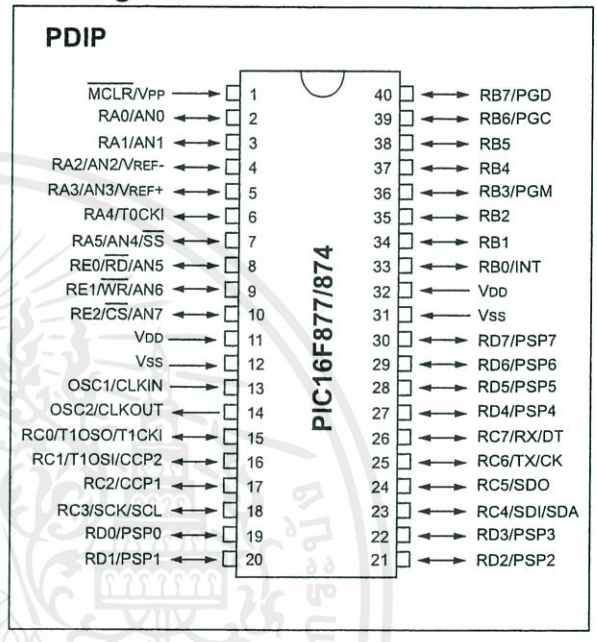
- PIC16F873
- PIC16F874
- PIC16F876
- PIC16F877

Microcontroller Core Features:

- High performance RISC CPU
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches which are two cycle
- Operating speed: DC - 20 MHz clock input
  DC - 200 ns instruction cycle
- Up to 8K x 14 words of FLASH Program Memory,
  Up to 368 x 8 bytes of Data Memory (RAM)
  Up to 256 x 8 bytes of EEPROM Data Memory
- Pinout compatible to the PIC16C73B/74B/76/77
- Interrupt capability (up to 14 sources)
- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- Power-on Reset (POR)
- Power-up Timer (PWRT) and
  Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection
- Power saving SLEEP mode
- Selectable oscillator options
- Low power, high speed CMOS FLASH/EEPROM technology
- Fully static design
- In-Circuit Serial Programming™ (ICSP) via two pins
- Single 5V In-Circuit Serial Programming capability
- In-Circuit Debugging via two pins
- Processor read/write access to program memory
- Wide operating voltage range: 2.0V to 5.5V
- High Sink/Source Current: 25 mA
- Commercial, Industrial and Extended temperature ranges
- Low-power consumption:
  - < 0.6 mA typical @ 3V, 4 MHz
  - 20 µA typical @ 3V, 32 kHz
  - < 1 µA typical standby current

Pin Diagram

PDIP

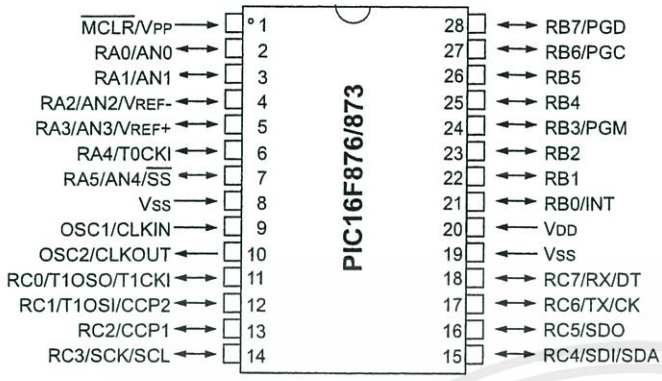


Peripheral Features:

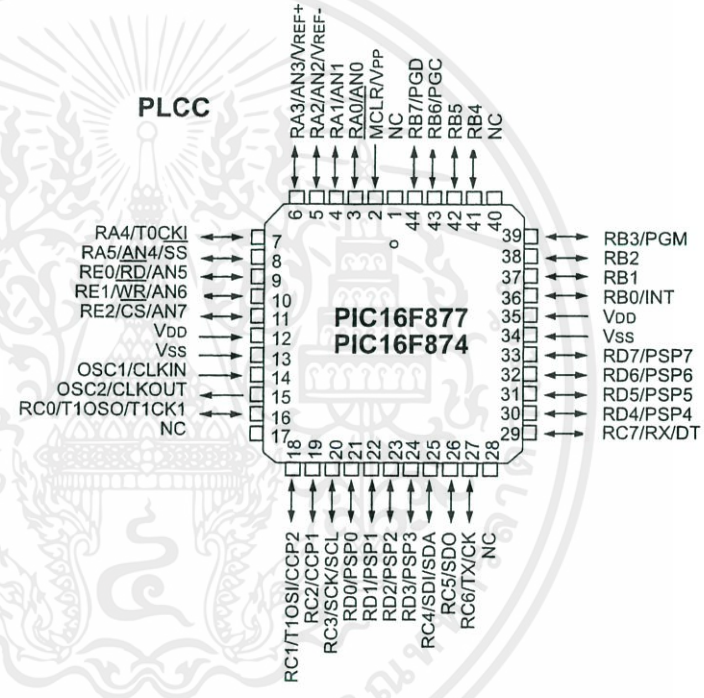
- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during SLEEP via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- 10-bit multi-channel Analog-to-Digital converter
- Synchronous Serial Port (SSP) with SPI™ (Master mode) and I^2C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) 8-bits wide, with external RD, WR and CS controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

Pin Diagrams

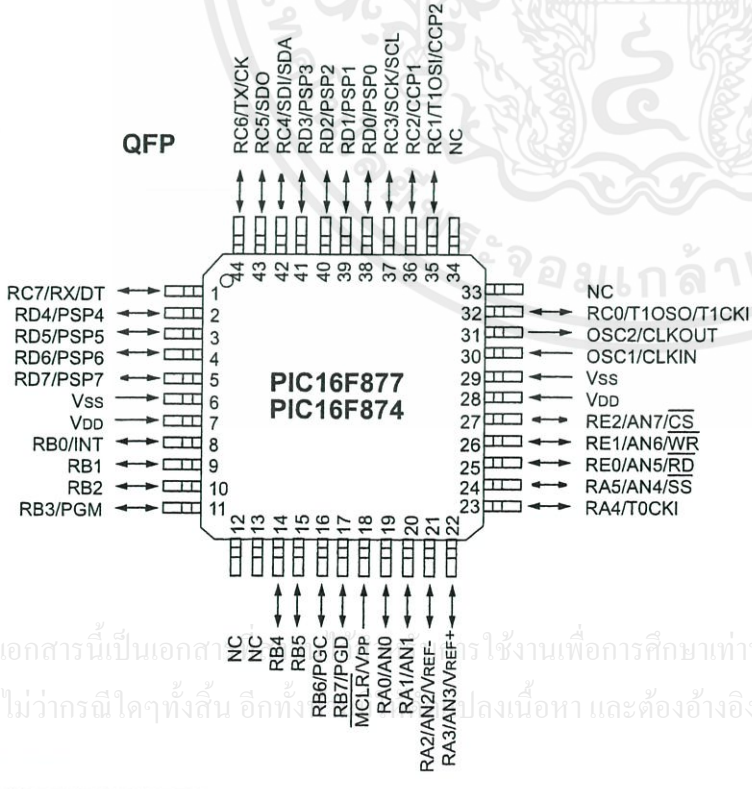
PDIP, SOIC



PLCC



QFP



Key Features PICmicro™ Mid-Range Reference Manual (DS33023)	PIC16F873	PIC16F874	PIC16F876	PIC16F877
Operating Frequency	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz
RESETS (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
FLASH Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory	128	128	256	256
Interrupts	13	14	13	14
I/O Ports	Ports A,B,C	Ports A,B,C,D,E	Ports A,B,C	Ports A,B,C,D,E
Timers	3	3	3	3
Capture/Compare/PWM Modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Instruction Set	35 instructions	35 instructions	35 instructions	35 instructions

Table of Contents

TO OUR VALUED CUSTOMERS

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your Microchip products. To this end, we will continue to improve our publications to better suit your needs. Our publications will be refined and enhanced as new volumes and updates are introduced.

If you have any questions or comments regarding this publication, please contact the Marketing Communications Department via E-mail at **docerrors@mail.microchip.com** or fax the **Reader Response Form** in the back of this data sheet to (480) 792-4150. We welcome your feedback.

Most Current Data Sheet

To obtain the most up-to-date version of this data sheet, please register at our Worldwide Web site at:

http://www.microchip.com

You can determine the version of a data sheet by examining its literature number found on the bottom outside corner of any page. The last character of the literature number is the version number, (e.g., DS30000A is version A of document DS30000).

Errata

An errata sheet, describing minor operational differences from the data sheet and recommended workarounds, may exist for current devices. As device/documentation issues become known to us, we will publish an errata sheet. The errata will specify the revision of silicon and revision of document to which it applies.

To determine if an errata sheet exists for a particular device, please check with one of the following:

- Microchip's Worldwide Web site; http://www.microchip.com
- Your local Microchip sales office (see last page)
- The Microchip Corporate Literature Center; U.S. FAX: (480) 792-7277

When contacting a sales office or the literature center, please specify which device, revision of silicon and data sheet (include literature number) you are using.

Customer Notification System

Register on our web site at **www.microchip.com/cn** to receive the most current information on all of our products.

1.0 DEVICE OVERVIEW

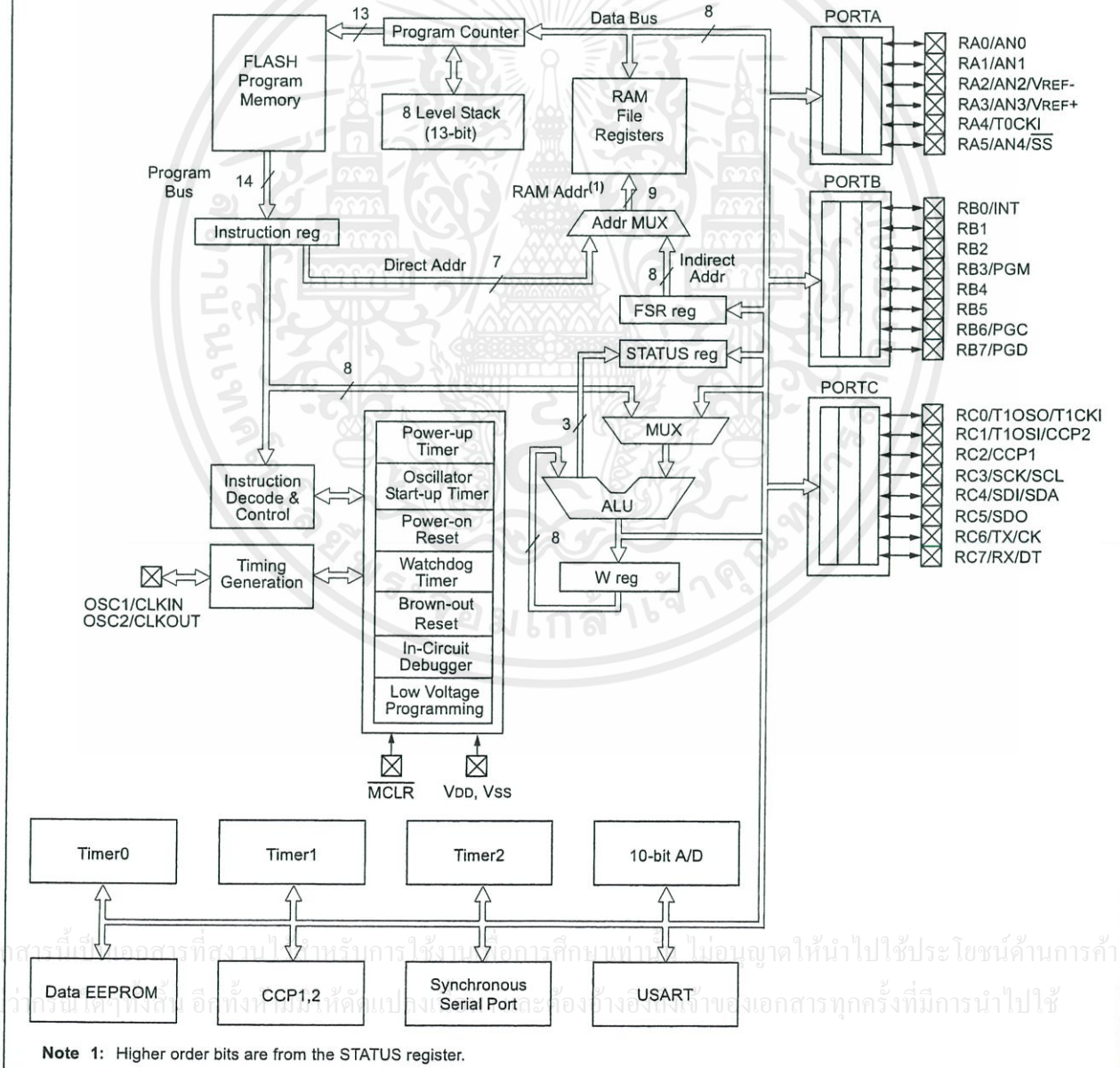
This document contains device specific information. Additional information may be found in the PICmicro™ Mid-Range Reference Manual (DS33023), which may be obtained from your local Microchip Sales Representative or downloaded from the Microchip website. The Reference Manual should be considered a complementary document to this data sheet, and is highly recommended reading for a better understanding of the device architecture and operation of the peripheral modules.

There are four devices (PIC16F873, PIC16F874, PIC16F876 and PIC16F877) covered by this data sheet. The PIC16F876/873 devices come in 28-pin packages and the PIC16F877/874 devices come in 40-pin packages. The Parallel Slave Port is not implemented on the 28-pin devices.

The following device block diagrams are sorted by pin number; 28-pin for Figure 1-1 and 40-pin for Figure 1-2. The 28-pin and 40-pin pinouts are listed in Table 1-1 and Table 1-2, respectively.

FIGURE 1-1: PIC16F873 AND PIC16F876 BLOCK DIAGRAM

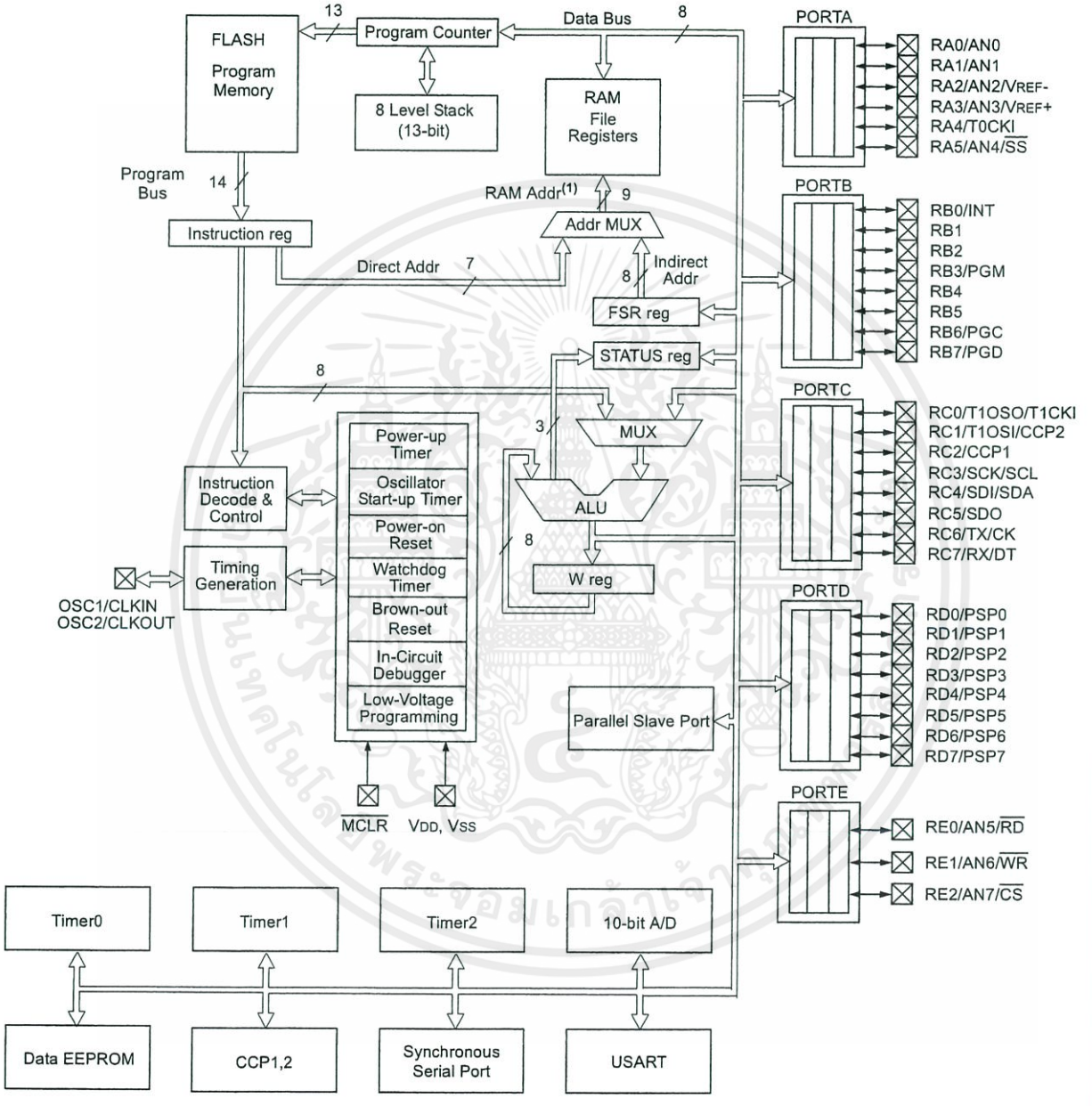
Device	Program FLASH	Data Memory	Data EEPROM
PIC16F873	4K	192 Bytes	128 Bytes
PIC16F876	8K	368 Bytes	256 Bytes



Note 1: Higher order bits are from the STATUS register.

FIGURE 1-2: PIC16F874 AND PIC16F877 BLOCK DIAGRAM

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874	4K	192 Bytes	128 Bytes
PIC16F877	8K	368 Bytes	256 Bytes



Note 1: Higher order bits are from the STATUS register.

TABLE 1-1: PIC16F873 AND PIC16F876 PINOUT DESCRIPTION

Pin Name	DIP Pin#	SOIC Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	9	9	I	ST/CMOS(3)	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	10	10	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, the OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
$\overline{MCLR}$/VPP	1	1	I/P	ST	Master Clear (Reset) input or programming voltage input. This pin is an active low RESET to the device.
					PORTA is a bi-directional I/O port.
RA0/AN0	2	2	I/O	TTL	RA0 can also be analog input0.
RA1/AN1	3	3	I/O	TTL	RA1 can also be analog input1.
RA2/AN2/VREF-	4	4	I/O	TTL	RA2 can also be analog input2 or negative analog reference voltage.
RA3/AN3/VREF+	5	5	I/O	TTL	RA3 can also be analog input3 or positive analog reference voltage.
RA4/T0CKI	6	6	I/O	ST	RA4 can also be the clock input to the Timer0 module. Output is open drain type.
RA5/$\overline{SS}$/AN4	7	7	I/O	TTL	RA5 can also be analog input4 or the slave select for the synchronous serial port.
					PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	21	21	I/O	TTL/ST(1)	RB0 can also be the external interrupt pin.
RB1	22	22	I/O	TTL	
RB2	23	23	I/O	TTL	
RB3/PGM	24	24	I/O	TTL	RB3 can also be the low voltage programming input.
RB4	25	25	I/O	TTL	Interrupt-on-change pin.
RB5	26	26	I/O	TTL	Interrupt-on-change pin.
RB6/PGC	27	27	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	28	28	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming data.
					PORTC is a bi-directional I/O port.
RC0/T1OSO/T1CKI	11	11	I/O	ST	RC0 can also be the Timer1 oscillator output or Timer1 clock input.
RC1/T1OSI/CCP2	12	12	I/O	ST	RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output.
RC2/CCP1	13	13	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/ PWM1 output.
RC3/SCK/SCL	14	14	I/O	ST	RC3 can also be the synchronous serial clock input/output for both SPI and I^2C modes.
RC4/SDI/SDA	15	15	I/O	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I^2C mode).
RC5/SDO	16	16	I/O	ST	RC5 can also be the SPI Data Out (SPI mode).
RC6/TX/CK	17	17	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	18	18	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.
VSS	8, 19	8, 19	P	—	Ground reference for logic and I/O pins.
VDD	20	20	P	—	Positive supply for logic and I/O pins.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

TABLE 1-2: PIC16F874 AND PIC16F877 PINOUT DESCRIPTION

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	13	14	30	I	ST/CMOS(4)	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	14	15	31	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
$\overline{MCLR}$/VPP	1	2	18	I/P	ST	Master Clear (Reset) input or programming voltage input. This pin is an active low RESET to the device.
						PORTA is a bi-directional I/O port.
RA0/AN0	2	3	19	I/O	TTL	RA0 can also be analog input0.
RA1/AN1	3	4	20	I/O	TTL	RA1 can also be analog input1.
RA2/AN2/VREF-	4	5	21	I/O	TTL	RA2 can also be analog input2 or negative analog reference voltage.
RA3/AN3/VREF+	5	6	22	I/O	TTL	RA3 can also be analog input3 or positive analog reference voltage.
RA4/T0CKI	6	7	23	I/O	ST	RA4 can also be the clock input to the Timer0 timer/ counter. Output is open drain type.
RA5/$\overline{SS}$/AN4	7	8	24	I/O	TTL	RA5 can also be analog input4 or the slave select for the synchronous serial port.
						PORTB is a bi-directional I/O port. PORTB can be soft-ware programmed for internal weak pull-up on all inputs.
RB0/INT	33	36	8	I/O	TTL/ST(1)	RB0 can also be the external interrupt pin.
RB1	34	37	9	I/O	TTL	
RB2	35	38	10	I/O	TTL	
RB3/PGM	36	39	11	I/O	TTL	RB3 can also be the low voltage programming input.
RB4	37	41	14	I/O	TTL	Interrupt-on-change pin.
RB5	38	42	15	I/O	TTL	Interrupt-on-change pin.
RB6/PGC	39	43	16	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	40	44	17	I/O	TTL/ST(2)	Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming data.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as an external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured as general purpose I/O and a TTL input when used in the Parallel Slave Port mode (for interfacing to a microprocessor bus).
4: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

TABLE 1-2: PIC16F874 AND PIC16F877 PINOUT DESCRIPTION (CONTINUED)

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O/P Type	Buffer Type	Description
						PORTC is a bi-directional I/O port.
RC0/T1OSO/T1CKI	15	16	32	I/O	ST	RC0 can also be the Timer1 oscillator output or a Timer1 clock input.
RC1/T1OSI/CCP2	16	18	35	I/O	ST	RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output.
RC2/CCP1	17	19	36	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	18	20	37	I/O	ST	RC3 can also be the synchronous serial clock input/ output for both SPI and I^2C modes.
RC4/SDI/SDA	23	25	42	I/O	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I^2C mode).
RC5/SDO	24	26	43	I/O	ST	RC5 can also be the SPI Data Out (SPI mode).
RC6/TX/CK	25	27	44	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	26	29	1	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.
						PORTD is a bi-directional I/O port or parallel slave port when interfacing to a microprocessor bus.
RD0/PSP0	19	21	38	I/O	ST/TTL(3)	
RD1/PSP1	20	22	39	I/O	ST/TTL(3)	
RD2/PSP2	21	23	40	I/O	ST/TTL(3)	
RD3/PSP3	22	24	41	I/O	ST/TTL(3)	
RD4/PSP4	27	30	2	I/O	ST/TTL(3)	
RD5/PSP5	28	31	3	I/O	ST/TTL(3)	
RD6/PSP6	29	32	4	I/O	ST/TTL(3)	
RD7/PSP7	30	33	5	I/O	ST/TTL(3)	
						PORTE is a bi-directional I/O port.
RE0/$\overline{RD}$/AN5	8	9	25	I/O	ST/TTL(3)	RE0 can also be read control for the parallel slave port, or analog input5.
RE1/$\overline{WR}$/AN6	9	10	26	I/O	ST/TTL(3)	RE1 can also be write control for the parallel slave port, or analog input6.
RE2/$\overline{CS}$/AN7	10	11	27	I/O	ST/TTL(3)	RE2 can also be select control for the parallel slave port, or analog input7.
VSS	12,31	13,34	6,29	P	—	Ground reference for logic and I/O pins.
VDD	11,32	12,35	7,28	P	—	Positive supply for logic and I/O pins.
NC	—	1,17,28, 40	12,13, 33,34		—	These pins are not internally connected. These pins should be left unconnected.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as an external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured as general purpose I/O and a TTL input when used in the Parallel Slave Port mode (for interfacing to a microprocessor bus).
4: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

CHAPTER 1

Instant Hacking: The Basics

It's time to start hacking.[1] In this chapter, you learn how to take control of your computer by speaking a language it understands: Python. Nothing here is particularly difficult, so if you know the basic principles of how your computer works, you should be able to follow the examples and try them out yourself. I'll go through the basics, startiwng with the excruciatingly simple, but because Python is such a powerful language, you'll soon be able to do pretty advanced things.

First, I show you how to get the software you need. Then I tell you a bit about algorithms and their main components. Throughout these sections, there are numerous small examples (most of them using only simple arithmetic) that you can try out in the Python interactive interpreter (covered in the section "The Interactive Interpreter" in this chapter). You learn about variables, functions, and modules, and after handling these topics, I show you how to write and run larger programs. Finally, I deal with strings, an important aspect of almost any Python program.

Installing Python

Before you can start programming, you need some new software. What follows is a short description of how to download and install Python. If you want to jump into the installation process without detailed guidance, you can simply visit `http://www.python.org/download` to get the most recent version of Python.

Windows

To install Python on a Windows machine, follow these steps:

1. Open a web browser and go to `http://www.python.org`.
2. Click the Download link.
3. You should see several links here, with names such as Python 2.5.*x* and Python 2.5.*x* Windows installer. Click the Windows installer link to download the installer file. (If you're running on an Itanium or AMD machine, you need to choose the appropriate installer.)

1. *Hacking* is not the same as *cracking*, which is a term describing computer crime. The two are often confused. Hacking basically means "having fun while programming." For more information, see Eric Raymond's article "How to Become a Hacker" at `http://www.catb.org/~esr/faqs/hacker-howto.html`.

Note If you can't find the link mentioned in step 3, click the link with the highest version among those with names like Python 2.5.*x*. For Python 2.5, you could simply go to `http://www.python.org/2.5`. Follow the instructions for Windows users. This will entail downloading a file called `python-2.5.x.msi` (or something similar), where 2.5.*x* should be the version number of the newest release.

4. Store the Windows Installer file somewhere on your computer, such as `C:\download\python-2.5.x.msi`. (Just create a directory where you can find it later.)
5. Run the downloaded file by double-clicking it in Windows Explorer. This brings up the Python install wizard, which is really easy to use. Just accept the default settings, wait until the installation is finished, and you're ready to roll!

Assuming that the installation went well, you now have a new program in your Windows Start menu. Run the Python Integrated Development Environment (IDLE) by selecting Start ➤ Programs ➤ Python[2] ➤ IDLE (Python GUI).

You should now see a window that looks like the one shown in Figure 1-1. If you feel a bit lost, simply select Help ➤ IDLE Help from the menu to get a simple description of the various menu items and basic usage. For more documentation on IDLE, check out `http://www.python.org/idle`. (Here you will also find more information on running IDLE on platforms other than Windows.) If you press F1, or select Help ➤ Python Docs from the menu, you will get the full Python documentation. (The document there of most use to you will probably be the Library Reference.) All the documentation is searchable.

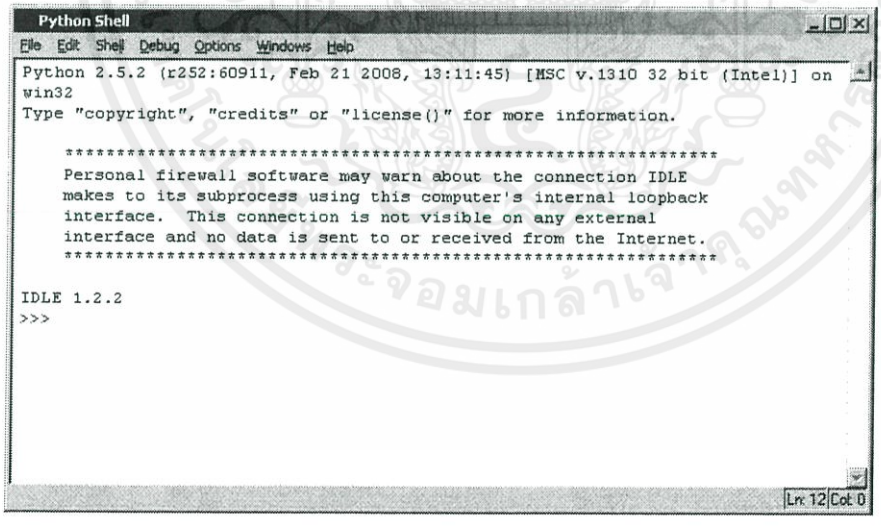


Figure 1-1. *The IDLE interactive Python shell*

2. This menu option will probably include your version number, as in Python 2.5.

Once you have the IDLE interactive Python shell running, you can continue with the section "The Interactive Interpreter," later in this chapter.

WINDOWS INSTALLER

Python for Microsoft Windows is distributed as a Windows Installer file, and requires that your Windows version supports Windows Installer 2.0 (or later). If you don't have Windows Installer, it can be downloaded freely for Windows 95, 98, ME, NT 4.0, and 2000. Windows XP and later versions of Windows already have Windows Installer, and many older machines will, too. There are download instructions for the Installer on the Python download page.

Alternatively, you could go to the Microsoft download site, `http://www.microsoft.com/downloads`, and search for "Windows Installer" (or simply select it from the download menu). Choose the most recent version for your platform and follow the download and installation instructions.

If you're uncertain about whether you have Windows Installer, simply try executing step 5 of the previous installation instructions: double-click the MSI file. If you get the install wizard, everything is okay. See `http://www.python.org/2.5/msi.html` for advanced features of the Windows Installer related to Python installation.

Linux and UNIX

In most Linux and UNIX installations (including Mac OS X), a Python interpreter will already be present. You can check whether this is the case for you by running the `python` command at the prompt, as follows:

```
$ python
```

Running this command should start the interactive Python interpreter, with output similar to the following:

```
Python 2.5.1 (r251:54869, Apr 18 2007, 22:08:04)
[GCC 4.0.1 (Apple Computer, Inc. build 5367)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

■ **Note** To exit the interactive interpreter, use Ctrl-D (press the Ctrl key and while keeping that depressed, press D).

If there is no Python interpreter installed, you will probably get an error message similar to the following:

```
bash: python: command not found
```

In that case, you need to install Python yourself, as described in the following sections.

Using a Package Manager

Several package systems and installation mechanisms exist for Linux. If you're running a Linux system with some form of package manager, chances are you can get Python through it.

■ **Note** You will probably need to have administrator privileges (a root account) in order to install Python using a package manager in Linux.

For example, if you're running Debian Linux, you should be able to install Python with the following command:

```
$ apt-get install python
```

If you're running Gentoo Linux, you should be able to use Portage, like this:

```
$ emerge python
```

In both cases, $ is, of course, the bash prompt.

■ **Note** Many other package managers out there have automatic download capabilities, including Yum, Synaptic (specific to Ubuntu Linux), and other Debian-style managers. You should probably be able to get recent versions of Python through these.

Compiling from Sources

If you don't have a package manager, or would rather not use it, you can compile Python yourself. This may be the method of choice if you are on a UNIX box but you don't have root access (installation privileges). This method is very flexible, and enables you to install Python wherever you want, including in your own home directory. To compile and install Python, follow these steps:

1. Go to the download page (refer to steps 1 and 2 in the instructions for installing Python on a Windows system).
2. Follow the instructions for downloading the sources.
3. Download the file with the extension `.tgz`. Store it in a temporary location. Assuming that you want to install Python in your home directory, you may want to put it in a directory such as `~/python`. Enter this directory (e.g., using `cd ~/python`).
4. Unpack the archive with the command `tar -xzvf Python-2.5.tgz` (where `2.5` is the version number of the downloaded source code). If your version of `tar` doesn't support the `z` option, you may want to uncompress the archive with `gunzip` first, and then use `tar -xvf` afterward. If there is something wrong with the archive, try downloading it again. Sometimes errors occur during download.

5. Enter the unpacked directory:

   ```
   $ cd Python-2.5
   ```

   Now you should be able to execute the following commands:

   ```
   ./configure --prefix=$(pwd)
   make
   make install
   ```

   You should end up with an executable file called `python` in the current directory. (If this doesn't work, consult the `README` file included in the distribution.) Put the current directory in your `PATH` environment variable, and you're ready to rock.

   To find out about the other configuration directives, execute this command:

   ```
   ./configure --help
   ```

Macintosh

If you're using a Macintosh with a recent version of Mac OS X, you'll have a version of Python installed already. Just open the Terminal application and enter the command `python` to start it. Even if you would like to install a newer version of Python, you should leave this one alone, as it is used in several parts of the operating system. You could use either MacPorts (`http://macports.org`) or Fink (`http://finkproject.org`), or you could use the distribution from the Python web site, by following these steps:

1. Go to the standard download page (see steps 1 and 2 from the Windows instructions earlier in this chapter).
2. Follow the link for the Mac OS X installer. There should also be a link to the MacPython download page, which has more information. The MacPython page also has versions of Python for older versions of the Mac OS.
3. Once you've downloaded the installer `.dmg` file, it will probably mount automatically. If not, simply double-click it. In the mounted disk image, you'll find an installer package (`.mpkg`) file. If you double-click this, the installation wizard will open, which will take you through the necessary steps.

Other Distributions

You now have the standard Python distribution installed. Unless you have a particular interest in alternative solutions, that should be all you need. If you are curious (and, perhaps, feeling a bit courageous), read on.

Several Python distributions are available in addition to the official one. The most well-known of these is probably ActivePython, which is available for Linux, Windows, Mac OS X, and several UNIX varieties. A slightly less well-known but quite interesting distribution is Stackless Python. These distributions are based on the standard implementation of Python, written in the C programming language. Two distributions that take a different approach are Jython and IronPython. If you're interested in development environments other than IDLE, Table 1-1 lists some options.

Table 1-1. *Some Integrated Development Environments (IDEs) for Python*

Environment	Description	Web Site
IDLE	The standard Python environment	http://www.python.org/idle
Pythonwin	Windows-oriented environment	http://www.python.org/download/windows
ActivePython	Feature-packed; contains Pythonwin IDE	http://www.activestate.com
Komodo	Commercial IDE	http://www.activestate.com[3]
Wingware	Commercial IDE	http://www.wingware.com
BlackAdder	Commercial IDE and (Qt) GUI builder	http://www.thekompany.com
Boa Constructor	Free IDE and GUI builder	http://boa-constructor.sf.net
Anjuta	Versatile IDE for Linux/UNIX	http://anjuta.sf.net
Arachno Python	Commercial IDE	http://www.python-ide.com
Code Crusader	Commercial IDE	http://www.newplanetsoftware.com
Code Forge	Commercial IDE	http://www.codeforge.com
Eclipse	Popular, flexible, open source IDE	http://www.eclipse.org
eric	Free IDE using Qt	http://eric-ide.sf.net
KDevelop	Cross-language IDE for KDE	http://www.kdevelop.org
VisualWx	Free GUI builder	http://visualwx.altervista.org
wxDesigner	Commercial GUI builder	http://www.roebling.de
wxGlade	Free GUI builder	http://wxglade.sf.net

ActivePython is a Python distribution from ActiveState (`http://www.activestate.com`). At its core, it's the same as the standard Python distribution for Windows. The main difference is that it includes a lot of extra goodies (modules) that are available separately. It's definitely worth a look if you are running Windows.

3. Komodo has been made open source, so free versions are also available.

Stackless Python is a reimplementation of Python, based on the original code, but with some important internal changes. To a beginning user, these differences won't matter much, and one of the more standard distributions would probably be more useful. The main advantages of Stackless Python are that it allows deeper levels of recursion and more efficient multithreading. As mentioned, both of these are rather advanced features, not needed by the average user. You can get Stackless Python from `http://www.stackless.com`.

Jython (`http://www.jython.org`) and IronPython (`http://www.codeplex.com/IronPython`) are different—they're versions of Python implemented in other languages. Jython is implemented in Java, targeting the Java Virtual Machine, and IronPython is implemented in C#, targeting the .NET and MONO implementations of the common language runtime (CLR). At the time of writing, Jython is quite stable, but lagging behind Python—the current Jython version is 2.2, while Python is at 2.5. There are significant differences in these two versions of the language. IronPython is still rather young, but it is quite usable, and it is reported to be faster than standard Python on some benchmarks.

Keeping in Touch and Up-to-Date

The Python language evolves continuously. To find out more about recent releases and relevant tools, the `python.org` web site is an invaluable asset. To find out what's new in a given release, go to the page for the given release, such as `http://python.org/2.5` for release 2.5. There you will also find a link to Andrew Kuchling's in-depth description of what's new for the release, with a URL such as `http://python.org/doc/2.5/whatsnew` for release 2.5. If there have been new releases since this book went to press, you can use these web pages to check out any new features.

Tip For a summary of what's changed in the more radically new release 3.0, see `http://docs.python.org/dev/3.0/whatsnew/3.0.html`.

If you want to keep up with newly released third-party modules or software for Python, check out the Python email list `python-announce-list`; for general discussions about Python, try `python-list`, but be warned: this list gets a *lot* of traffic. Both of these lists are available at `http://mail.python.org`. If you're a Usenet user, these two lists are also available as the newsgroups `comp.lang.python.announce` and `comp.lang.python`, respectively. If you're totally lost, you could try the `python-help` list (available from the same place as the two other lists) or simply email `help@python.org`. Before you do, you really ought to see if your question is a frequently asked one, by consulting the Python FAQ, at `http://python.org/doc/faq`, or by performing a quick Web search.

The Interactive Interpreter

When you start up Python, you get a prompt similar to the following:

```
Python 2.5.1 (r251:54869, Apr 18 2007, 22:08:04)
[GCC 4.0.1 (Apple Computer, Inc. build 5367)] on darwin
```

```
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

■ **Note** The exact appearance of the interpreter and its error messages will depend on which version you are using.

This might not seem very interesting, but believe me—it is. This is your gateway to hackerdom—your first step in taking control of your computer. In more pragmatic terms, it's an interactive Python interpreter. Just to see if it's working, try the following:

```
>>> print "Hello, world!"
```

When you press the Enter key, the following output appears:

```
Hello, world!
>>>
```

■ **Note** If you are familiar with other computer languages, you may be used to terminating every line with a semicolon. There is no need to do so in Python. A line is a line, more or less. You may add a semicolon if you like, but it won't have any effect (unless more code follows on the same line), and it is not a common thing to do.

What happened here? The >>> thingy is the prompt. You can write something in this space, like `print "Hello, world!"`. If you press Enter, the Python interpreter prints out the string "Hello, world!" and you get a new prompt below that.

■ **Note** The term "printing" in this context refers to writing text to the screen, not producing hard copies with a printer.

What if you write something completely different? Try it out:

```
>>> The Spanish Inquisition
SyntaxError: invalid syntax
>>>
```

Obviously, the interpreter didn't understand that.[4] (If you are running an interpreter other than IDLE, such as the command-line version for Linux, the error message will be slightly

4. After all, no one expects the Spanish Inquisition . . .

dfferent.) The interpreter also indicates what's wrong: it will emphasize the word *Spanish* by giving it a red background (or, in the command-line version, by using a caret, ^).

If you feel like it, play around with the interpreter some more. For some guidance, try entering the command `help` at the prompt and pressing Enter. As mentioned, you can press F1 for help about IDLE. Otherwise, let's press on. After all, the interpreter isn't much fun when you don't know what to tell it, is it?

Algo . . . What?

Before you start programming in earnest, I'll try to give you an idea of what computer programming is. Simply put, it's telling a computer what to do. Computers can do a lot of things, but they aren't very good at thinking for themselves. They really need to be spoon-fed the details. You need to feed the computer an algorithm in some language it understands. *Algorithm* is just a fancy word for a procedure or recipe—a detailed description of how to do something. Consider the following:

```
SPAM with SPAM, SPAM, Eggs, and SPAM:
First, take some SPAM.
Then add some SPAM, SPAM, and eggs.
If a particularly spicy SPAM is desired, add some SPAM.
Cook until done - Check every 10 minutes.
```

This recipe may not be very interesting, but how it's constructed is. It consists of a series of instructions to be followed in order. Some of the instructions may be done directly ("take some SPAM"), while some require some deliberation ("If a particularly spicy SPAM is desired"), and others must be repeated several times ("Check every 10 minutes.")

Recipes and algorithms consist of ingredients (objects, things), and instructions (statements). In this example, SPAM and eggs were the ingredients, while the instructions consisted of adding SPAM, cooking for a given length of time, and so on. Let's start with some reasonably simple Python ingredients and see what you can do with them.

Numbers and Expressions

The interactive Python interpreter can be used as a powerful calculator. Try the following:

```
>>> 2 + 2
```

This should give you the answer 4. That wasn't too hard. Well, what about this:

```
>>> 53672 + 235253
288925
```

Still not impressed? Admittedly, this is pretty standard stuff. (I'll assume that you've used a calculator enough to know the difference between `1+2*3` and `(1+2)*3`.) All the usual arithmetic operators work as expected—almost. There is one potential trap here, and that is integer division (in Python versions prior to 3.0):

```
>>> 1/2
0
```

What happened here? One integer (a nonfractional number) was divided by another, and the result was rounded down to give an integer result. This behavior can be useful at times, but often (if not most of the time), you need ordinary division. What do you do to get that? There are two possible solutions: use real numbers (numbers with decimal points) rather than integers, or tell Python to change how division works.

Real numbers are called *floats* (or *floating-point numbers*) in Python. If either one of the numbers in a division is a float, the result will be, too:

```
>>> 1.0 / 2.0
0.5

>>> 1/2.0
0.5
>>> 1.0/2
0.5

>>> 1/2.
0.5
```

If you would rather have Python do proper division, you could add the following statement to the beginning of your program (writing full programs is described later) or simply execute it in the interactive interpreter:

```
>>> from __future__ import division
```

■ **Note** In case it's not entirely clear, the `future` in the instruction is surrounded by two underscores on both sides: `__future__`.

Another alternative, if you're running Python from the command line (e.g., on a Linux machine), is to supply the command-line switch `-Qnew`. In either case, division will suddenly make a bit more sense:

```
>>> 1 / 2
0.5
```

Of course, the single slash can no longer be used for the kind of integer division shown earlier. A separate operator will do this for you—the double slash:

```
>>> 1 // 2
0
```

The double slash consistently performs integer division, even with floats:

```
>>> 1.0 // 2.0
0.0
```

There is a more thorough explanation of the `__future__` stuff in the section "Back to the `__future__`," later in this chapter.

Now you've seen the basic arithmetic operators (addition, subtraction, multiplication, and division), but one more operator is quite useful at times:

```
>>> 1 % 2
1
```

This is the remainder (modulus) operator. `x % y` gives the remainder of `x` divided by `y`. Here are a few more examples:

```
>>> 10 / 3
3
>>> 10 % 3
1
>>> 9 / 3
3
>>> 9 % 3
0
>>> 2.75 % 0.5
0.25
```

Here `10/3` is `3` because the result is rounded down. But 3 × 3 is 9, so you get a remainder of 1. When you divide 9 by 3, the result is exactly 3, with no rounding. Therefore, the remainder is 0. This may be useful if you want to check something "every 10 minutes" as in the recipe earlier in the chapter. You can simply check whether `minute % 10` is 0. (For a description on how to do this, see the sidebar "Sneak Peek: The `if` Statement," later in this chapter.) As you can see from the final example, the remainder operator works just fine with floats as well.

The last operator is the exponentiation (or power) operator:

```
>>> 2 ** 3
8
>>> -3 ** 2
-9
>>> (-3) ** 2
9
```

Note that the exponentiation operator binds tighter than the negation (unary minus), so `-3**2` is in fact the same as `-(3**2)`. If you want to calculate `(-3)**2`, you must say so explicitly.

Large Integers

Python can handle really large integers:

```
>>> 10000000000000000000
10000000000000000000L
```

What happened here? The number suddenly got an `L` tacked onto the end.

Note If you're using a version of Python older than 2.2, you get the following behavior:

```
>>> 1000000000000000000
OverflowError: integer literal too large
```

The newer versions of Python are more flexible when dealing with big numbers.

Ordinary integers can't be larger than 2147483647 (or smaller than –2147483648). If you want really big numbers, you must use longs. A *long* (or *long integer*) is written just like an ordinary integer but with an `L` at the end. (You can, in theory, use a lowercase `l` as well, but that looks all too much like the digit 1, so I'd advise against it.)

In the previous example, Python converted the integer to a long, but you can do that yourself, too. Let's try that big number again:

```
>>> 1000000000000000000L
1000000000000000000L
```

Of course, this is only useful in old versions of Python that aren't capable of figuring this stuff out.

Well, can you do math with these monster numbers, too? Sure thing. Consider the following:

```
>>> 1987163987163981639186L * 198763981726391826L + 23
394976626432005567613000143784791693659L
```

As you can see, you can mix long integers and plain integers as you like. In all likelihood, you won't have to worry about the difference between longs and ints unless you're doing type checking, as described in Chapter 7—and that's something you should almost never do.

Hexadecimals and Octals

To conclude this section, I should mention that hexadecimal numbers are written like this:

```
>>> 0xAF
175
```

and octal numbers like this:

```
>>> 010
8
```

The first digit in both of these is zero. (If you don't know what this is all about, just close your eyes and skip to the next section—you're not missing anything important.)

Note For a summary of Python's numeric types and operators, see Appendix B.

Variables

Another concept that might be familiar to you is *variables*. If math makes you queasy, don't worry: variables in Python are easy to understand. A variable is basically a name that represents (or refers to) some value. For example, you might want the name x to represent 3. To make it so, simply execute the following:

```
>>> x = 3
```

This is called an *assignment*. We assign the value 3 to the variable x. Another way of putting this is to say that we bind the variable x to the value (or object) 3. After you've assigned a value to a variable, you can use the variable in expressions:

```
>>> x * 2
6
```

Note that you need to assign a value to a variable before you use it. After all, it doesn't make any sense to use a variable if it doesn't represent a value, does it?

■ **Note** Variable names can consist of letters, digits, and underscore characters (_). A variable can't begin with a digit, so `Plan9` is a valid variable name, whereas `9Plan` is not.

Statements

Until now we've been working (almost) exclusively with expressions, the ingredients of the recipe. But what about statements—the instructions?

In fact, I've cheated. I've introduced two types of statements already: the `print` statement, and assignments. So, what's the difference between a statement and an expression? Well, an expression *is* something, while a statement *does* something (or, rather, tells the computer to do something). For example, `2*2` *is* 4, whereas `print 2*2` *prints* 4. What's the difference? After all, they behave very similarly. Consider the following:

```
>>> 2*2
4
>>> print 2*2
4
```

As long as you execute this in the interactive interpreter, the results are similar, but that is only because the interpreter always prints out the values of all expressions (using the same representation as `repr`—see the section "String Representations, str and repr" later in this chapter). That is not true of Python in general. Later in this chapter, you'll see how to make programs that run without this interactive prompt, and simply putting an expression such as `2*2` in your program won't do anything interesting.[5] Putting `print 2*2` in there, on the other hand, will print out 4.

---

5. In case you're wondering—yes, it *does* do something. It calculates the product of 2 and 2. However, the result isn't kept anywhere or shown to the user; it has no *side effects*, beyond the calculation itself.

Note In Python 3.0, `print` is a function, which means you need to write `print(42)` instead of `print 42`, for example. Other than that, it works more or less like the statement, as described here.

The difference between statements and expressions may be more obvious when dealing with assignments. Because they are not expressions, they have no values that can be printed out by the interactive interpreter:

```
>>> x = 3
>>>
```

As you can see, you get a new prompt immediately. Something has changed, however; `x` is now bound to the value `3`.

This is a defining quality of statements in general: they change things. For example, assignments change variables, and `print` statements change how your screen looks.

Assignments are, perhaps, the most important type of statement in any programming language, although it may be difficult to grasp their importance right now. Variables may just seem like temporary "storage" (like the pots and pans of a cooking recipe), but the real power of variables is that you don't need to know what values they hold in order to manipulate them.[6] For example, you know that `x * y` evaluates to the product of `x` and `y`, even though you may have no knowledge of what `x` and `y` are. So, you may write programs that use variables in various ways without knowing the values they will eventually hold (or refer to) when the program is run.

Getting Input from the User

You've seen that you can write programs with variables without knowing their values. Of course, the interpreter must know the values eventually. So how can it be that we don't? The interpreter knows only what we tell it, right? Not necessarily.

You may have written a program, and someone else may use it. You cannot predict what values users will supply to the program. Let's take a look at the useful function `input`. (I'll have more to say about functions in a minute.)

```
>>> input("The meaning of life: ")
The meaning of life: 42
42
```

What happens here is that the first line (`input(...)`) is executed in the interactive interpreter. It prints out the string `"The meaning of life: "` as a new prompt. I type `42` and press

6. Note the quotes around storage. Values aren't stored in variables—they're stored in some murky depths of computer memory, and are referred to by variables. As will become abundantly clear as you read on, more than one variable can refer to the same value.

Enter. The resulting value of `input` is that very number, which is automatically printed out in the last line. That may not seem very useful, but look at the following:

```
>>> x = input("x: ")
x: 34
>>> y = input("y: ")
y: 42
>>> print x * y
1428
```

Here, the statements at the Python prompts (`>>>`) could be part of a finished program, and the values entered (`34` and `42`) would be supplied by some user. Your program would then print out the value `1428`, which is the product of the two. And you didn't have to know these values when you wrote the program, right?

Note This is much more useful when you save your programs in a separate file so other users can execute them. You learn to do that later in this chapter, in the section "Saving and Executing Your Programs."

SNEAK PEEK: THE IF STATEMENT

To make things a bit more fun, I'll give you a sneak peek of something you aren't really supposed to learn about until Chapter 5: the `if` statement. The `if` statement lets you perform an action (another statement) if a given condition is true. One type of condition is an equality test, using the equality operator `==`. Yes, it's a *double* equality sign. The single one is used for assignments, remember?

You simply put this condition after the word `if` and then separate it from the following statement with a colon:

```
>>> if 1 == 2: print 'One equals two'
...
>>> if 1 == 1: print 'One equals one'
...
One equals one
>>>
```

As you can see, nothing happens when the condition is false. When it is true, however, the following statement (in this case, a `print` statement) is executed. Note also that when using `if` statements in the interactive interpreter, you need to press Enter twice before it is executed. (The reason for this will become clear in Chapter 5—don't worry about it for now.)

So, if the variable `time` is bound to the current time in minutes, you could check whether you're "on the hour" with the following statement:

```
if time % 60 == 0: print 'On the hour!'
```

Functions

In the section on numbers and expressions, I used the exponentiation operator (**) to calculate powers. The fact is that you can use a *function* instead, called pow:

```
>>> 2**3
8
>>> pow(2,3)
8
```

A function is like a little program that you can use to perform a specific action. Python has a lot of functions that can do many wonderful things. In fact, you can make your own functions, too (more about that later); therefore, we often refer to standard functions such as pow as *built-in* functions.

Using a function as I did in the preceding example is called *calling* the function. You supply it with *parameters* (in this case, 2 and 3) and it *returns* a value to you. Because it returns a value, a function call is simply another type of *expression*, like the arithmetic expressions discussed earlier in this chapter.[7] In fact, you can combine function calls and operators to create more complicated expressions:

```
>>> 10 + pow(2, 3*5)/3.0
10932.666666666666
```

Note The exact number of decimals may vary depending on which version of Python you are using.

Several built-in functions can be used in numeric expressions like this. For example, abs gives the absolute value of a number, and round rounds floating-point numbers to the nearest integer:

```
>>> abs(-10)
10
>>> 1/2
0
>>> round(1.0/2.0)
1.0
```

Notice the difference between the two last expressions. Integer division always rounds down, whereas round rounds to the nearest integer. But what if you want to round a given number down? For example, you might know that a person is 32.9 years old, but you would like to round that down to 32 because she isn't really 33 yet. Python has a function for this (called floor)—it just isn't available directly. As is the case with many useful functions, it is found in a *module*.

7. Function calls can also be used as statements if you simply ignore the return value.

Modules

You may think of modules as extensions that can be imported into Python to extend its capabilities. You import modules with a special command called (naturally enough) `import`. The function mentioned in the previous section, `floor`, is in a module called `math`:

```
>>> import math
>>> math.floor(32.9)
32.0
```

Notice how this works: we import a module with `import`, and then use the functions from that module by writing `module.function`.

If you want the age to be an integer (`32`) and not a float (`32.0`), you can use the function `int`:[8]

```
>>> int(math.floor(32.9))
32
```

Note Similar functions exist to convert to other types (for example, `long` and `float`). In fact, these aren't completely normal functions—they're *type objects*. I'll have more to say about types later. The opposite of `floor` is `ceil` (short for "ceiling"), which finds the smallest integral value larger than or equal to the given number.

If you are sure that you won't import more than one function with a given name (from different modules), you might not want to write the module name each time you call the function. Then you can use a variant of the `import` command:

```
>>> from math import sqrt
>>> sqrt(9)
3.0
```

After using `from module import function`, you can use the function without its module prefix.

Tip You may, in fact, use variables to refer to functions (and most other things in Python). For example, by performing the assignment `foo = math.sqrt`, you can start using `foo` to calculate square roots; for example, `foo(4)` yields `2.0`.

---

8. The `int` function/type will actually round down while converting to an integer, so when converting to an integer, using `math.floor` is superfluous; you could simply use `int(32.9)`.

cmath and Complex Numbers

The sqrt function is used to calculate the square root of a number. Let's see what happens if we supply it with a negative number:

```
>>> from math import sqrt
>>> sqrt(-1)
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in ?
    sqrt(-1)
ValueError: math domain error
```

or, on some platforms:

```
>>> sqrt(-1)
nan
```

Note nan is simply a special value meaning "not a number."

Well, that's reasonable. You can't take the square root of a negative number—or can you? Indeed you can. The square root of a negative number is an imaginary number. (This is a standard mathematical concept—if you find it a bit too mind-bending, feel free to skip ahead.) So why couldn't sqrt deal with it? Because it deals only with floats, and imaginary numbers (and complex numbers, the sum of real and imaginary numbers) are something completely different—which is why they are covered by a different module, cmath (for complex math):

```
>>> import cmath
>>> cmath.sqrt(-1)
1j
```

Notice that I didn't use from ... import ... here. If I had, I would have lost my ordinary sqrt. Name clashes like these can be sneaky, so unless you really want to use the from version, you should probably stick with a plain import.

The value 1j is an imaginary number. These numbers are written with a trailing j (or J), just like longs use L. Without delving into the theory of complex numbers, let me just show a final example of how you can use them:

```
>>> (1+3j) * (9+4j)
(-3+31j)
```

As you can see, the support for complex numbers is built into the language.

Note There is no separate type for imaginary numbers in Python. They are treated as complex numbers whose real component is zero.

Back to the __future__

It has been rumored that Guido van Rossum (Python's creator) has a time machine, because quite often when people request features in the language, the features have already been implemented. Of course, we aren't all allowed into this time machine, but Guido has been kind enough to build a part of it into Python, in the form of the magic module `__future__`. From it, we can import features that will be standard in Python in the future but that aren't part of the language yet. You saw this in the section about numbers and expressions, and you'll be bumping into it from time to time throughout this book.

Saving and Executing Your Programs

The interactive interpreter is one of Python's great strengths. It makes it possible to test solutions and to experiment with the language in real time. If you want to know how something works, just try it! However, everything you write in the interactive interpreter is lost when you quit. What you really want to do is write programs that both you and other people can run. In this section, you learn how to do just that.

First of all, you need a text editor, preferably one intended for programming. (If you use something like Microsoft Word, which I don't really recommend, be sure to save your code as plain text.) If you are already using IDLE, you're in luck. With IDLE, you can simply create a new editor window with File ➤ New Window. Another window appears, without an interactive prompt. Whew!

Start by entering the following:

```
print "Hello, world!"
```

Now select File ➤ Save to save your program (which is, in fact, a plain text file). Be sure to put it somewhere where you can find it later. You might want to create a directory where you put all your Python projects, such as `C:\python` in Windows. In a UNIX environment, you might use a directory like `~/python`. Give your file any reasonable name, such as `hello.py`. The `.py` ending is important.

Note If you followed the installation instructions earlier in this chapter, you may have put your Python installation in `~/python` already, but because that has a subdirectory of its own (such as `~/python/Python-2.5/`), this shouldn't cause any problems. If you would rather put your own programs somewhere else, feel free to use a directory such as `~/my_python_programs`.

Got that? Don't close the window with your program in it. If you did, just open it again (File ➤ Open). Now you can run it with Edit ➤ Run script, or by pressing Ctrl+F5. (If you aren't using IDLE, see the next section about running your programs from the command prompt.)

What happens? `Hello, world!` is printed in the interpreter window, which is exactly what we wanted. The interpreter prompt may be gone (depending on the version you're using), but you can get it back by pressing Enter (in the interpreter window).

Let's extend our script to the following:

```
name = raw_input("What is your name? ")
print "Hello, " + name + "!"
```

■ **Note** Don't worry about the difference between `input` and `raw_input`—I'll get to that.

If you run this (remember to save it first), you should see the following prompt in the interpreter window:

```
What is your name?
```

Enter your name (for example, `Gumby`) and press Enter. You should get something like this:

```
Hello, Gumby!
```

Fun, isn't it?

Running Your Python Scripts from a Command Prompt

Actually, there are several ways to run your programs. First, let's assume that you have a DOS window or a UNIX shell prompt before you, and that the directory containing the Python executable (called `python.exe` in Windows, and `python` in UNIX) or the directory *containing* the executable (in Windows) has been put in your `PATH` environment variable.[9] Also, let's assume that your script from the previous section (`hello.py`) is in the current directory. Then you can execute your script with the following command in Windows:

```
C:\>python hello.py
```

or UNIX:

```
$ python hello.py
```

As you can see, the command is the same. Only the system prompt changes.

■ **Note** If you don't want to mess with environment variables, you can simply specify the full path of the Python interpreter. In Windows, you might do something like this:

```
C:\>C:\Python25\python hello.py
```

Making Your Scripts Behave Like Normal Programs

Sometimes you want to execute a Python program (also called a *script*) the same way you execute other programs (such as your web browser or text editor), rather than explicitly using the

9. If you don't understand this sentence, you should perhaps skip the section. You don't really need it.

Python interpreter. In UNIX, there is a standard way of doing this: have the first line of your script begin with the character sequence `#!` (called *pound bang* or *shebang*) followed by the absolute path to the program that interprets the script (in our case Python). Even if you didn't quite understand that, just put the following in the first line of your script if you want it to run easily on UNIX:

```
#!/usr/bin/env python
```

This should run the script, regardless of where the Python binary is located.

■ **Note** In some operating systems if you install a recent version of Python (e.g., 2.5) you will still have an old one lying around (e.g.,1.5.2), which is needed by some system programs (so you can't uninstall it). In such cases, the `/usr/bin/env` trick is not a good idea, as you will probably end up with your programs being executed by the old Python. Instead, you should find the exact location of your new Python executable (probably called `python` or `python2`) and use the full path in the pound bang line, like this:

```
#!/usr/bin/python2
```

The exact path may vary from system to system.

Before you can actually run your script, you must make it executable:

```
$ chmod a+x hello.py
```

Now it can be run like this (assuming that you have the current directory in your path):

```
$ hello.py
```

If this doesn't work, try using `./hello.py` instead, which will work even if the current directory (`.`) is not part of your execution path.

If you like, you can rename your file and remove the `py` suffix to make it look more like a normal program.

What About Double-Clicking?

In Windows, the suffix (`.py`) is the key to making your script behave like a program. Try double-clicking the file `hello.py` you saved in the previous section. If Python was installed correctly, a DOS window appears with the prompt "What is your name?" Cool, huh?[10] (You'll see how to make your programs look better, with buttons, menus, and so on, later.)

There is one problem with running your program like this, however. Once you've entered your name, the program window closes before you can read the result. The window closes when the program is finished. Try changing the script by adding the following line at the end:

```
raw_input("Press <enter>")
```

10. This behavior depends on your operating system and the installed Python interpreter. If you've saved the file using IDLE in Mac OS X, for example, double-clicking the file will simply open it in the IDLE code editor.

Now, after running the program and entering your name, you should have a DOS window with the following contents:

```
What is your name? Gumby
Hello, Gumby!
Press <enter>
```

Once you press the Enter key, the window closes (because the program is finished). Just as a teaser, rename your file `hello.pyw`. (This is Windows-specific.) Double-click it as before. What happens? Nothing! How can that be? I will tell you later in the book—I promise.

Comments

The hash sign (#) is a bit special in Python. When you put it in your code, everything to the right of it is ignored (which is why the Python interpreter didn't choke on the `/usr/bin/env` stuff used earlier). Here is an example:

```
# Print the circumference of the circle:
print 2 * pi * radius
```

The first line here is called a *comment*, which can be useful in making programs easier to understand—both for other people and for yourself when you come back to old code. It has been said that the first commandment of programmers is "Thou Shalt Comment" (although some less charitable programmers swear by the motto "If it was hard to write, it should be hard to read"). Make sure your comments say significant things and don't simply restate what is already obvious from the code. Useless, redundant comments may be worse than none. For example, in the following, a comment isn't really called for:

```
# Get the user's name:
user_name = raw_input("What is your name?")
```

It's always a good idea to make your code readable on its own as well, even without the comments. Luckily, Python is an excellent language for writing readable programs.

Strings

Now what was all that `raw_input` and `"Hello, " + name + "!"` stuff about? Let's tackle the `"Hello"` part first and leave `raw_input` for later.

The first program in this chapter was simply

```
print "Hello, world!"
```

It is customary to begin with a program like this in programming tutorials. The problem is that I haven't really explained how it works yet. You know the basics of the `print` statement (I'll have more to say about that later), but what is `"Hello, world!"`? It's called a *string* (as in "a string of characters"). Strings are found in almost every useful, real-world Python program and have many uses. Their main use is to represent bits of text, such as the exclamation "Hello, world!"

Single-Quoted Strings and Escaping Quotes

Strings are values, just as numbers are:

```
>>> "Hello, world!"
'Hello, world!'
```

There is one thing that may be a bit surprising about this example, though: when Python printed out our string, it used single quotes, whereas we used double quotes. What's the difference? Actually, there is no difference:

```
>>> 'Hello, world!'
'Hello, world!'
```

Here, we use single quotes, and the result is the same. So why allow both? Because in some cases it may be useful:

```
>>> "Let's go!"
"Let's go!"
>>> '"Hello, world!" she said'
'"Hello, world!" she said'
```

In the preceding code, the first string contains a single quote (or an apostrophe, as we should perhaps call it in this context), and therefore we can't use single quotes to enclose the string. If we did, the interpreter would complain (and rightly so):

```
>>> 'Let's go!'
SyntaxError: invalid syntax
```

Here, the string is `'Let'`, and Python doesn't quite know what to do with the following `s` (or the rest of the line, for that matter).

In the second string, we use double quotes as part of our sentence. Therefore, we have to use single quotes to enclose our string, for the same reasons as stated previously. Or, actually we don't *have* to. It's just convenient. An alternative is to use the backslash character (`\`) to escape the quotes in the string, like this:

```
>>> 'Let\'s go!'
"Let's go!"
```

Python understands that the middle single quote is a character *in* the string and not the *end* of the string. (Even so, Python chooses to use double quotes when printing out the string.) The same works with double quotes, as you might expect:

```
>>> "\"Hello, world!\" she said"
'"Hello, world!" she said'
```

Escaping quotes like this can be useful, and sometimes necessary. For example, what would you do without the backslash if your string contained both single and double quotes, as in the string `'Let\'s say "Hello, world!"'`?

Note Tired of backslashes? As you will see later in this chapter, you can avoid most of them by using long strings and raw strings (which can be combined).

Concatenating Strings

Just to keep whipping this slightly tortured example, let me show you another way of writing the same string:

```
>>> "Let's say " '"Hello, world!"'
'Let\'s say "Hello, world!"'
```

I've simply written two strings, one after the other, and Python automatically concatenates them (makes them into one string). This mechanism isn't used very often, but it can be useful at times. However, it works only when you actually write both strings at the same time, directly following one another:

```
>>> x = "Hello, "
>>> y = "world!"
>>> x y
SyntaxError: invalid syntax
```

In other words, this is just a special way of writing strings, not a general method of concatenating them. How, then, do you concatenate strings? Just like you add numbers:

```
>>> "Hello, " + "world!"
'Hello, world!'
>>> x = "Hello, "
>>> y = "world!"
>>> x + y
'Hello, world!'
```

String Representations, str and repr

Throughout these examples, you have probably noticed that all the strings printed out by Python are still quoted. That's because it prints out the value as it might be written in Python code, not how you would like it to look for the user. If you use `print`, however, the result is different:

```
>>> "Hello, world!"
'Hello, world!'
>>> 10000L
10000L
>>> print "Hello, world!"
Hello, world!
>>> print 10000L
10000
```

As you can see, the long integer `10000L` is simply the number 10000 and should be written that way when presented to the user. But when you want to know what value a variable refers to, you may be interested in whether it's a normal integer or a long, for example.

What is actually going on here is that values are converted to strings through two different mechanisms. You can use both mechanisms yourself, through the functions `str` and `repr`. `str` simply converts a value into a string in some reasonable fashion that will probably be understood by a user, for example.[11] `repr` creates a string that is a representation of the value as a legal Python expression. Here are a few examples:

```
>>> print repr("Hello, world!")
'Hello, world!'
>>> print repr(10000L)
10000L
>>> print str("Hello, world!")
Hello, world!
>>> print str(10000L)
10000
```

A synonym for `repr(x)` is `` `x` `` (here, you use backticks, not single quotes). This can be useful when you want to print out a sentence containing a number:

```
>>> temp = 42
>>> print "The temperature is " + temp
Traceback (most recent call last):
  File "<pyshell#61>", line 1, in ?
    print "The temperature is " + temp
TypeError: cannot add type "int" to string
>>> print "The temperature is " + `temp`
The temperature is 42
```

Note Backticks are removed in Python 3.0, so even though you may find backticks in old code, you should probably stick with `repr` yourself.

The first `print` statement doesn't work because you can't add a string to a number. The second one, however, works because I have converted `temp` to the string `"42"` by using the backticks. (I could have just as well used `repr`, which means the same thing, but may be a bit clearer. Actually, in this case, I could also have used `str`. Don't worry too much about this right now.)

In short, `str`, `repr`, and backticks are three ways of converting a Python value to a string. The function `str` makes it look good, while `repr` (and the backticks) tries to make the resulting string a legal Python expression.

11. Actually, `str` is a type, just like `int` and `long`. `repr`, however, is simply a function.

input vs. raw_input

Now you know what `"Hello, " + name + "!"` means. But what about `raw_input`? Isn't `input` good enough? Let's try it. Enter the following in a separate script file:

```
name = input("What is your name? ")
print "Hello, " + name + "!"
```

This is a perfectly valid program, but as you will soon see, it's a bit impractical. Let's try to run it:

```
What is your name? Gumby
Traceback (most recent call last):
  File "C:/python/test.py", line 2, in ?
    name = input("What is your name? ")
  File "<string>", line 0, in ?
NameError: name 'Gumby' is not defined
```

The problem is that `input` assumes that what you enter is a valid Python expression (it's more or less the inverse of `repr`). If you write your name as a string, that's no problem:

```
What is your name? "Gumby"
Hello, Gumby!
```

However, it's just a bit too much to ask that users write their name in quotes like this. Therefore, we use `raw_input`, which treats all input as raw data and puts it into a string:

```
>>> input("Enter a number: ")
Enter a number: 3
3
>>> raw_input("Enter a number: ")
Enter a number: 3
'3'
```

Unless you have a special need for `input`, you should probably use `raw_input`.

Long Strings, Raw Strings, and Unicode

Before ending this chapter, I want to tell you about a few other ways of writing strings. These alternate string syntaxes can be useful when you have strings that span several lines or contain various special characters.

Long Strings

If you want to write a really long string, one that spans several lines, you can use triple quotes instead of ordinary quotes:

```
print '''This is a very long string.
It continues here.
And it's not over yet.
"Hello, world!"
Still here.'''
```

You can also use triple double quotes, `"""like this"""`. Note that because of the distinctive enclosing quotes, both single and double quotes are allowed inside, without being backslash-escaped.

Tip Ordinary strings can also span several lines. If the last character on a line is a backslash, the line break itself is "escaped" and ignored. For example:

```
print "Hello, \
world!"
```

would print out `Hello, world!`. The same goes for expressions and statements in general:

```
>>> 1 + 2 + \
       4 + 5
12
>>> print \
      'Hello, world'
Hello, world
```

Raw Strings

Raw strings aren't too picky about backslashes, which can be very useful sometimes.[12] In ordinary strings, the backslash has a special role: it *escapes* things, letting you put things into your string that you couldn't normally write directly. For example, a new line is written `\n`, and can be put into a string like this:

```
>>> print 'Hello,\nworld!'
Hello,
world!
```

This is normally just dandy, but in some cases, it's not what you want. What if you wanted the string to include a backslash followed by an `n`? You might want to put the DOS pathname `C:\nowhere` into a string:

```
>>> path = 'C:\nowhere'
>>> path
'C:\nowhere'
```

This looks correct, until you print it and discover the flaw:

```
>>> print path
C:
owhere
```

12. Raw strings can be especially useful when writing regular expressions. More about those in Chapter 10.

Not exactly what we were after, is it? So what do we do? We can escape the backslash itself:

```
>>> print 'C:\\nowhere'
C:\nowhere
```

This is just fine. But for long paths, you wind up with a lot of backslashes:

```
path = 'C:\\Program Files\\fnord\\foo\\bar\\baz\\frozz\\bozz'
```

Raw strings are useful in such cases. They don't treat the backslash as a special character at all. Every character you put into a raw string stays the way you wrote it:

```
>>> print r'C:\nowhere'
C:\nowhere
>>> print r'C:\Program Files\fnord\foo\bar\baz\frozz\bozz'
C:\Program Files\fnord\foo\bar\baz\frozz\bozz
```

As you can see, raw strings are prefixed with an `r`. It would seem that you can put anything inside a raw string, and that is almost true. Quotes must be escaped as usual, although that means that you get a backslash in your final string, too:

```
>>> print r'Let\'s go!'
Let\'s go!
```

The one thing you can't have in a raw string is a lone, final backslash. In other words, the last character in a raw string cannot be a backslash unless you escape it (and then the backslash you use to escape it will be part of the string, too). Given the previous example, that ought to be obvious. If the last character (before the final quote) is an unescaped backslash, Python won't know whether or not to end the string:

```
>>> print r"This is illegal\"
SyntaxError: invalid token
```

Okay, so it's reasonable, but what if you want the last character in your raw string to be a backslash? (Perhaps it's the end of a DOS path, for example.) Well, I've given you a whole bag of tricks in this section that should help you solve that problem, but basically you need to put the backslash in a separate string. A simple way of doing that is the following:

```
>>> print r'C:\Program Files\foo\bar' '\\'
C:\Program Files\foo\bar\
```

Note that you can use both single and double quotes with raw strings. Even triple-quoted strings can be raw.

Unicode Strings

The final type of string constant is the *Unicode string* (or Unicode *object*—they don't really belong to the same type as strings). If you don't know what Unicode is, you probably don't need to know about this. (If you want to find out more about it, you can go to the Unicode web site, `www.unicode.org`.) Normal strings in Python are stored internally as 8-bit ASCII, while Unicode strings are stored as 16-bit Unicode. This allows for a more varied set of characters,

including special characters from most languages in the world. I'll restrict my treatment of Unicode strings to the following:

```
>>> u'Hello, world!'
u'Hello, world!'
```

As you can see, Unicode strings use the prefix `u`, just as raw strings use the prefix `r`.

■ **Note** In Python 3.0, all strings will be Unicode strings.

A Quick Summary

This chapter covered quite a bit of material. Let's take a look at what you've learned before moving on.

Algorithms: An algorithm is a recipe telling you exactly how to perform a task. When you program a computer, you are essentially describing an algorithm in a language the computer can understand, such as Python. Such a machine-friendly description is called a program, and it mainly consists of expressions and statements.

Expressions: An expression is a part of a computer program that represents a value. For example, `2+2` is an expression, representing the value `4`. Simple expressions are built from *literal values* (such as `2` or `"Hello"`) by using *operators* (such as `+` or `%`) and *functions* (such as `pow`). More complicated expressions can be created by combining simpler expressions (e.g., `(2+2)*(3-1)`). Expressions may also contain *variables*.

Variables: A variable is a name that represents a value. New values may be assigned to variables through *assignments* such as `x = 2`. An assignment is a kind of *statement*.

Statements: A statement is an instruction that tells the computer to *do* something. That may involve changing variables (through assignments), printing things to the screen (such as `print "Hello, world!"`), importing modules, or a host of other stuff.

Functions: Functions in Python work just like functions in mathematics: they may take some arguments, and they return a result. (They may actually do lots of interesting stuff before returning, as you will find out when you learn to write your own functions in Chapter 6.)

Modules: Modules are extensions that can be imported into Python to extend its capabilities. For example, several useful mathematical functions are available in the `math` module.

Programs: You have looked at the practicalities of writing, saving, and running Python programs.

Strings: Strings are really simple—they are just pieces of text. And yet there is a lot to know about them. In this chapter, you've seen many ways to write them, and in Chapter 3 you learn many ways of using them.

New Functions in This Chapter

Function	Description
`abs(number)`	Returns the absolute value of a number
`cmath.sqrt(number)`	Returns the square root; works with negative numbers
`float(object)`	Converts a string or number to a floating-point number
`help()`	Offers interactive help
`input(prompt)`	Gets input from the user
`int(object)`	Converts a string or number to an integer
`long(object)`	Converts a string or number to a long integer
`math.ceil(number)`	Returns the ceiling of a number as a float
`math.floor(number)`	Returns the floor of a number as a float
`math.sqrt(number)`	Returns the square root; doesn't work with negative numbers
`pow(x, y[, z])`	Returns x to the power of y (modulo z)
`raw_input(prompt)`	Gets input from the user, as a string
`repr(object)`	Returns a string representation of a value
`round(number[, ndigits])`	Rounds a number to a given precision
`str(object)`	Converts a value to a string

What Now?

Now that you know the basics of expressions, let's move on to something a bit more advanced: data structures. Instead of dealing with simple values (such as numbers), you'll see how to bunch them together in more complex structures, such as lists and dictionaries. In addition, you'll take another close look at strings. In Chapter 5, you learn more about statements, and after that you'll be ready to write some really nifty programs.