**บทที่ 2**

**เอกสารและงานวิจัยที่เกี่ยวข้อง**

ในการศึกษาเรื่องแขนกลหยิบจับสิ่งของควบคุมด้วยไมโครคอนโทรลเลอร์ ได้รวบรวมแนวคิดทฤษฎีและหลักการต่างๆจากเอกสารและงานวิจัยที่เกี่ยวข้องดังต่อไปนี้

1. เซอร์โวมอเตอร์

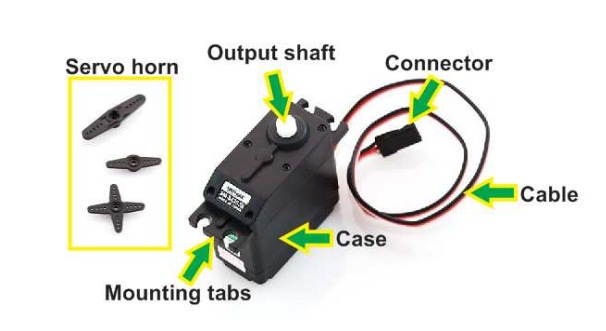
2. บอร์ด อะดูโน่

3. โครงสร้างโปรแกรมของ Arduino

**2.1. เซอร์โวมอเตอร์**

servo เป็นคําศัพท์ที่ใช้กันทั่วไปในระบบควบคุมอัตโนมัติมาจากภาษาละตินคําว่าServoหมายถึง“ทาส” (Slave) ในเชิงความหมายของ Servo Motor ก็คือ Motor ที่เราสามารถสั่งงานหรือตั้งค่า แล้วตัว Motor จะหมุนไปยังตําแหน่งองศาที่เราสั่งได้เองอย่างถูกต้อง โดยใช้การควบคุมแบบป้อนกลับ(Feedback Control) ในบทความนี้จะกล่าวถึง RC Servo Motor ซึ่งนิยมนํามาใช้ในเครื่องเล่นที่บังคับด้วยคลื่นวิทยุ (RC = Radio - Controlled) เช่น เรือบังคับวิทยุ รถบังคับวิทยุ เฮลิคอปเตอร์บังคับวิทยุ เป็นต้น

Feedback Control คือ ระบบควบคุมที่มีการวัดค่าเอาต์พุตของระบบนํามาเปรียบเทียบกับค่าอินพุตเพื่อควบคุมและปรับแต่งให้ค่าเอาต์พุตของระบบให้มีค่า เท่ากับ หรือ ใกล้เคียงกับค่าอินพุต



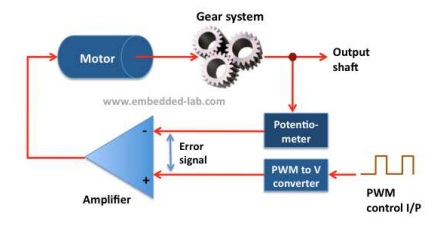
**รูปที่ 2.1.** ส่วนประกอบภายนอก RC Servo Moto

- Case ตัวถัง หรือ กรอบของตัว Servo Motor  
 - Mounting Tab ส่วนจับยึดตัว Servo กับชิ้นงาน  
 - Output Shaft เพลาส่งกําลัง  
 - Servo Horns ส่วนเชื่อมต่อกับ Output shaft เพื่อสร้างกลไกล  
 - Cable สายเชื่อมต่อเพื่อ จ่ายไฟฟ้า และ ควบคุม Servo Motor จะประกอบด้วยสายไฟ 3 เส้น และ ใน RC Servo Motor จะมีสีของสายแตกต่างกันไปดังนี้  
- สายสีแดง คือ ไฟเลี้ยง (4.8-6V)  
- สายสีดํา หรือ น้ําตาล คือ กราวด์



**รูปที่ 2.2.** ส่วนประกอบภายใน RC Servo Motor

1. Motor เป็นส่วนของตัวมอเตอร์  
2. Gear Train หรือ Gearbox เป็นชุดเกียร์ทดแรง  
3. Position Sensor เป็นเซ็นเซอร์ตรวจจับตําแหน่งเพื่อหาค่าองศาในการหมุน  
4. Electronic Control System เป็นส่วนที่ควบคุมและประมวลผล

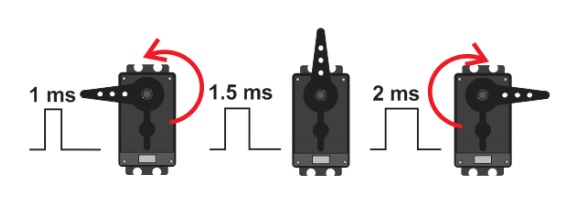


**รูปที่ 2.3.** Servo Motor Block Diagram

ตามภาพด้านบนซึ่งแสดงระบบการทํางานของเซอร์โวโดยที่รีซีฟจะส่งสัญญาณการควบคุม  
ตําแหน่งของเซอร์โวไปยังส่วน Control Circuit ของเซอร์โวโดยสัญญาณที่ส่งมาจะเป็นสัญญาณแบบPWM(Pule Width Modulation) จากนั้น Control Circuit จะถอดรหัสสัญญาณPWM ที่ได้ให้เป็นตําแหน่งของเซอร์โวที่ถูกต้องโดยเปรียบเทียบค่าตําแหน่งปัจจุบันกับสัญญาณกลับจาPotentiometerแล้วจึงส่งแรงดันไฟฟ้าไปยังมอเตอร์ให้ไปหมุนไปในทิศทางที่จะทําให้ตําแหน่งของPotentiometer มีค่าที่ถูกต้องเท่ากับค่าที่ได้ถอดรหัสมาซึ่งขณะที่มอเตอร์หมุนก็จะมีเฟืองที่ไปต่อกับแกPotentiometer  
(ปรกติจะอยู่ในแกนเดียวกับoutputshaft)ด้วยดังนั้นกระบวนการนี้จะเกิดขึ้น ช้าๆจนกว่าค่าของPotentiometer จะมีค่าเท่ากับการถอดรหัสสัญญาณที่ได้รับมาจากรีซีฟการทํางานของมอเตอร์จึงจะหยุดแต่กระบวนการทํางานของControlCircuitจะยังทํางานอยู่ตลอดเวลาเพียงแต่หากค่าของPotentiometer มีค่าเท่ากับสัญญาณที่ถอดรหัสมาจากรีซีฟแล้วก็จะไม่มีการส่งแรงดันไฟฟ้าไปยังมอเตอร์(ไม่มีการเปลี่ยนแปลงของสติ๊กที่รีโมท) ซึ่งกระบวนการนี้เรียกว่าการอัพเดทสัญญาณโดยมีความเร็วที่ 50ครั้งต่อหนึ่งวินาที เราจึงเห็นเป็นการเคลื่อนที่ของเซอร์โว

**สัญญาณ RC ในรูปแบบ PWM**

ตัว RC Servo Motor ออกแบบมาใช้สําหรับรับคําสั่งจาก Remote Control ที่ใช้ควบคุมของเล่นด้วยสัญญาณวิทยุต่างๆ เช่น เครื่องบินบังคับ รถบังบังคับ เรือบังคับ เป็นต้น ซึ่ง Remote จําพวกนี้ที่ภาครับจะแปลงความถี่วิทยุออกมาในรูปแบบสัญญาณ PWM (Pulse Width Modulation)มุมหรือองศาจะขึ้นอยู่กับความกว้างของสัญญาณพัลซ์ ซึ่งโดยส่วนมากความกว้างของพัลซ์ที่ใช้ใน RC Servo Motorจะอยู่ในช่วง 1-2 ms หรือ 0.5-2.5 msยกตัวอย่างเช่นหากกําหนดความกว้างของสัญญาณพัลซ์ไว้ที่ 1 ms ตัว Servo Motor จะหมุนไปทางด้าย



**รูปที่ 2.4.** ความกว้างของสัญญาณพัลซ์แต่ละองศา

ดังนั้นสามารถกําหนดองศาการหมุนของ RC Servo Motor ได้โดยการเทียบค่า เช่น RC ServoMotor สามารถหมุนได้ 180 องศา โดยที่ 0 องศาใช้ความกว้างพัลซ์เท่ากับ 1000 us ที่ 180 องศาความกว้างพัลซ์เท่ากับ 2000 us เพราะฉะนั้นค่าที่เปลี่ยนไป 1 องศาจะใช้ความกว้างพัลซ์ต่างกัน (2000-1000)/180 เท่ากับ 5.55 us  
 จากการหาค่าความกว้างพัลซ์ที่มุม 1 องศาข้างต้น หากต้องกําหนดให้ RC Servo Motor หมุนไปที่มุม 45 องศาจะหาค่าพัลซ์ที่ต้องการได้จาก 5.55 x 45 เท่ากับ 249.75 us แต่ที่มุม 0 องศาเราเริ่มที่ความกว้างพัลซ์ 1ms หรือ 1000 us เพราะฉะนั้นความกว้างพัลซ์ที่ใช้กําหนดให้ RC Servo Motor หมุนไปที่ 45 องศา คือ 1000 + 249.75 เท่ากับประมาณ 1250 us

**Speed และ Torque Ratings**  
 นอกเหนือจากขนาดภายนอกของเซอร์โวที่เราต้องพิจารณาในการใช้งานแล้วยังมีคุณลักษณะที่เราต้องพิจารณาอีกก็คือ speed(ความเร็ว) และ Torque (แรงบิด)Speed การวัดความเร็วของเซอร์โวก็คือเวลาที่เซอร์โวใช้ต่อองศาในการหมุนค่าหนึ่ง ซึ่งมุมมาตรฐานที่ใช้วัดกันทั่วไปคือ 60 องศา จึงกล่าวได้ว่า ความเร็วของเซอร์โวก็คือ เวลาที่ใช้ในการหมุนแขนของเซอร์โวไปจากตําแหน่งเดิมเป็นมุม 60 องศา ดังนั้น ตัวเลขเวลาที่มีค่าน้อยเท่าไหร่ หมายถึงเซอร์โวยิ่งความเร็วมากขึ้นเท่านั้น  
ยกตัวอย่างเช่น เซอร์โวตัวหนึ่งมีความเร็ว 0.12 sec/60° ซึ่งหมายถึงเซอร์โวตัวนี้ใช้เวลา 0.12 วินาที ในการหมุนแขนของเซอร์โวเป็นมุม 60 องศา ซึ่งถือว่าเป็นมีความเร็วเป็นสองเท่าของความเร็วของเซอร์โวมาตรฐานซึ่งมีความเร็วอยู่ที่ 0.24 sec/60° แต่สําหรับเซอร์โวที่ใช้ควบคุมหางของเฮลิคอปเตอร์วิทยุบังคับแล้วอาจต้องใช้เซอร์โวที่มีความเร็วที่มากขึ้นเช่น 0.06 sec/60°  
 Torque แรงบิด หมายถึง แรงที่แขนของเซอร์โวสามารถกระทําได้ ซึ่งแรงนี้วัดกันในหน่วยของออนซ์ ต่อ นิ้ว(oz-in) หรือ กิโลกรัม ต่อ เซนติเมตร (kg-cm) ซึ่งตัวเลขที่มากหมายถึงแรงที่ที่เซอร์โวสามารถฉุดหรือผลักได้ โดยที่ standard servo แรงบิดจะอยู่ที่ประมาณ 40 oz-in สําหรับเซอร์โวประเภทแรงบิดสูง (high torque) อาจมีแรงบิดสูงถึง 200 oz-in แล้ว 40 ounce-inches หมายถึงอะไร หมายถึง ถ้าแขนของเซอร์โวมีความยาวหนึ่งนิ้วจะมีแรงดึงหรือแรงฉุดขนาด 40 ounces ก่อนที่เซอร์โวจะมาสามรถหมุนได้อีก (Stalling) ดังนั้นหากแขนของเซอร์โวมีความยาก ½ นิ้ว ก็จะสามารถดึงหรือฉุดได้ 80 ounces และ หากแขนของเซอร์โวยาว 2 นิ้ว ก็จะสามารถฉุดหรือดึงได้ 20 ounces ตามหลักการคํานวนโมเมนต์  
 ความเร็วและแรงบิดของเซอร์โวจะมีค่าต่างกันเมื่อจ่ายแรงดันไฟฟ้าที่ต่างกันให้กับรีซีฟ โดยทั่วไป จะมีอยู่ 2 ค่าคือ 4.8 โวลท์ และ 6.0 โวลท์ ซึ่งจะได้มาจากการจ่ายแรงดันไฟฟ้าของ BEC’s หรือ อุปกรณ์ปรับแรงดันไฟฟ้า (voltage regulator) และแน่นอนอยู่แล้วว่าแรงดันไฟฟ้าสูงกว่าย่อมให้ ความเร็วและแรงบิดที่สูงกว่า จึงได้มีเซอร์โวที่รองรับแรงดันไฟฟ้าได้ถึง 8.6 volts ซึ่งให้ ความเร็วและแรงบิดที่สูงมากทั้งยังสามารถรับแรงดันไฟฟ้าจากแบ็ต 2S Lipo ได้โดยไม่ต้องมีวงจรปรับแรงดันไฟฟ้าและแน่นอนว่ารีซีฟก็ต้องสามารถรองรับแรงดันไฟฟ้านี้ได้ด้วยซึ่งรีซีฟระบบ 2.4 GhZ ในปัจจุบันส่วนใหญ่ก็สามารถรองรับได้ส่วนองค์ประกอบที่ทําให้ต้องจํากัดของแรงดันไฟฟ้าสําหรับเฮลิคอปเตอร์ที่นอกเหนือจากเครื่องบินคือไจโร (Gyro) และ Gyro servo ซึ่งส่วนมากจะออกแบบมาให้ทํางานกับแรงดันไฟฟ้าไม่เกิน 5 โวลท์ แต่ แนวโน้มในปัจจุบันก็ได้มีการผลิต Gyro ที่สามารถทํางานได้กับแรงดันไฟฟ้าที่สูงขึ้นMetal Gears & Metal Output Shafts:  
 ชนิดของเฟืองก็เป็นอีกปัจจัยหนึ่งในการเลือกเซอร์โวซึ่งมีอยู่ด้วยกัน 3 ประเภทหลักๆคือ



รูปที่ 2.5. Nylon หรือที่เรียกันว่าเฟืองพลาสติก

Nylon หรือที่เรียกกันว่าเฟืองพลาสติกซึ่งแน่นอนว่าความแข็งแรงนั้นค่อนข้างน้อยแต่ประโยชน์คือ มีน้ำหนักน้อยทําให้ตอบสนองได้ไวกว่าเฟืองที่มีน้ำหนักมาก



**รูปที่ 2.6.** Metal หรือที่เรียกกันว่าเฟืองโลหะ

Metal หรือที่เรียกกันว่าเฟืองโลหะซึ่งจะให้ความแข็งแรงที่มากกว่าแต่ก็แลกมาด้วยน้ำหนักที่มาก และมีการตอบสนองช้ากว่าโลหะที่ใช้ส่วนใหญ่ก็เป็นทองเหลืองแต่สําหรับเครื่องบินบางแบบที่ต้องการ น้ำหนักเบาด้วยก็อาจจะทําจากไททาเนียม



**รูปที่ 2.7.** การผสมระหว่างเฟืองโลหะครึ่งหนึ่งและเฟืองพลาสติกครึ่งหนึ่ง

Metal ½ หรือ Nylon ½ นั่นคือมีการผสมระหว่างเฟืองโลหะครึ่งหนึ่งและเฟืองพลาสติกครึ่งหนึ่งเพื่อเป็นการเอาข้อดีของแต่ละแบบมาผสมกันแต่นั่นก็ขึ้นอยู่กับการใช้งาน



**รูปที่ 2.8.** ตัวอย่างสเป็คของเซอร์โวค่าย Towerpro

**2.1.1. วิธีควบคุม RC Servo Motor ด้วย Arduino**  
 Arduino มีไลบรารี่สําหรับสั่งงาน RC Servo Motor มาให้ใช้งานอยู่แล้วเป็นฟังก์ชั่นสําเร็จรูปและใช้งานได้ง่ายใน ได้ให้ข้อมูลไว้ว่า Servo Library ของ Arduino สามารถสั่งงาน RC ServoMotor ได้ทั้งแบบหมุนไป-กลับได้ 0-180 องศา และแบบต่อเนื่องที่หมุนครบรอบได้เรียกว่าเป็นContinuous Rotation Servo โดยสามารถรองรับการเชื่อมต่อ RC Servo Motor ได้ถึง 12 ตัวกับบอร์ดArduino UNO และรองรับสูงสุดถึง 48 ตัวหากใช้บอร์ด Arduino Mega  
**ฟังก์ชั่นภายใน Servo Library**  
 - attach()  
 - write()  
 - writeMicroseconds()  
 - read()  
 - attached()  
 - detach()  
 **attach()**  
 Description  
 คือฟังก์ชั่นที่ใช้ในการกําหนดขาสัญญาณที่ Servo Motor ต่อกับ Arduino และกําหนดความ  
กว้างของพัลซ์ที่ 0 องศาและ 180 องศา  
 Syntax  
 Servo.attach(pin)  
 Servo.attach(pin,min,max)  
 Parameters  
 Pin คือ ขาสัญญาณของ Arduino ที่ใช้เชื่อมต่อกับ Servo Motor  
 Min คือ ความกว้างของพัลซ์ที่ 0 องศาของ Servo ตัวที่ใช้ในหน่วยไมโครวินาที (us) โดยปกติ  
แล้วหากไม่มีการตั้งค่าโปรแกรมจะกําหนดค่าไว้ที่ 544 us  
 Max คือ ความกว้างของพัลซ์ที่ 180 องศาของ Servo ตัวที่ใช้ในหน่วยไมโครวินาที (us) โดยปกติแล้วหากไม่มีการตั้งค่าโปรแกรมจะกําหนดค่าไว้ที่ 2400 us

**Write()** Description  
 คือฟังก์ชั่นที่ใช้ควบคุมตําแหน่งที่ต้องการให้ Servo Motor หมุนไปยังองศาที่กําหนดสามารถ  
กําหนดเป็นค่าองศาได้เลย คือ 0-180 องศา แต่ใน Servo Motor ที่เป็น Full Rotation คําสั่ง write จะเป็นการกําหนดความเร็วในการหมุน โดย  
 ค่าเท่ากับ 90 คือคําสั่งให้ Servo Motor หยุดหมุน  
 ค่าเท่ากับ 0 คือการหมุนด้วยความเร็วสูงสุดในทิศทางหนึ่ง  
 ค่าเท่ากับ 180 คือการหมุนด้วยความเร็วสูงสุดในทิศทางตรงกันข้าม  
 Syntax  
 servo.write(angle)  
 Parameters  
 Angle คือมุมที่ต้องการให้ RC Servo Motor แบบ 0-180 องศาหมุนไป แต่หากเป็น RC Servo Motor แบบ Full Rotation ค่า Angle คือ การกําหนดความเร็วและทิศทางในการหมุน **writeMicroseconds()**  
 Description  
 คือฟังก์ชั่นที่ใช้ควบคุมตําแหน่งที่ให้ Servo Motor หมุนไปยังตําแหน่งองศาที่กําหนดโดยกําหนดเป็นค่าความกว้างของพัลซ์ในหน่วย us ซึ่งปกติแล้ว RC Servo Motor จะใช้ความกว้างของพัลซ์อยู่ที่1000-2000 us ตามที่ได้กล่าวไปข้างต้นแล้ว แต่ RC Servo Motor บางรุ่นหรือบางยี่ห้อไม่ได้ใช้ ช่วงความกว้างของพัลซ์ตามที่ได้กล่าวเอาไว้นี้ อาจจะใช้ช่วง 700-2300 แทนก็สามารถใช้ฟังก์ชั่นwriteMicroseconds นี้เพื่อกําหนดความกว้างพัลซ์ได้เองการใช้ฟังก์ชั่นwriteMicroseconds สามารถกําหนดค่าได้อิสระ ตรงนี้ ”ต้องระวังในการใช้งาน”  
หากสั่งงาน RC Servo Motor (แบบ 0 - 180 องศา) จนหมุนไปเกินจุดสิ้นสุดคือเกินทั้งฝั่ง 0 หรือ 180 องศา จะทําให้เกิดเสียงครางดังจากการหมุนไปต่อไม่ได้และมอเตอร์จะกินกระแสสูงขึ้นด้วยในเวลาเดียวกันนั้น ซึ่งอาจทําให้ RC Servo Motor เกิดความเสียหายได้ Syntax  
 servo.writeMicroseconds(uS)  
Parameters  
 uS คือค่าความกว้างของพัลซ์ที่ต้องการกําหนดในหน่วยไมโครวินาที (โดยตัวแปร int)  
 **read()**  
 Description  
 คือฟังก์ชั่นอ่านค่าองศาที่สั่งเข้าไปด้วยฟังก์ชั่น write() เพื่อให้รู้ว่าตําแหน่งองศาสุดท้ายที่เราสั่งเข้าไปนั้นมีค่าเท่าไหร่ซึ่งค่าที่อ่านออกมานั้นจะมีค่าอยู่ในช่วง 0 - 180  
 Syntax  
 **servo.read()**  
 Parameters  
 ไม่มี จะ Return ค่า 0-180  
**attached()**  
 Description  
 คือฟังก์ชั่นตรวจสอบว่า Servo ที่เราต้องการใช้กําลังต่ออยู่กับขสัญญาณของ Arduino หรือไม่  
 Syntax  
**servo.attached()**  
 Parameters  
 ไม่มี จะ Return ค่า True ออกมา หาก Servo Motor เชื่อมต่ออยู่กับ Arduino แต่ถ้าหาก Return ออกมาเป็นค่าอื่นถือว่าไม่เชื่อมต่อ

**detach()**  
 Description  
 คือฟังก์ชั่นคืนสถานะของขาที่เรากําหนดให้เป็นขาควบคุม Servo Motor ด้วยคําสั่งattached() ให้กลับคือสู่การใช่งานปกติ  
Syntaxservo.detach()Parametersไม่มีโค้ดตัวอย่างการควบคุมตําแหน่ง RC Servo Motor

#include <Servo.h>  
Servo myservo;  
void setup()  
{  
myservo.attach(9);  
}

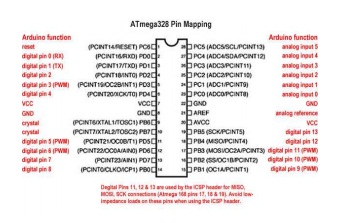
void loop()  
{  
myservo.write(0);  
11  
delay(1000);  
myservo.write(90);  
delay(1000);  
myservo.write(180);  
delay(1000);  
}  
ผลการทํางานของโค้ดmyservo.write(0);delay(1000);  
Servo Motor จะหมุนไปที่ตําแหน่ง 0 องศา และ หยุดเป็นเวลา 1 วินาทีmyservo.write(90);delay(1000);  
Servo Motor จะหมุนไปที่ตําแหน่ง 90 องศา และ หยุดเป็นเวลา 1 วินาทีmyservo.write(90);delay(1000);  
Servo Motor จะหมุนไปที่ตําแหน่ง 180 องศา และ หยุดเป็นเวลา 1 วินาทีจากนั้นจะหมุนกลับไปที่ตําแหน่ง 0 องศา และวนรอบไปเช่นนี้เรื่อยๆ

**2.2. บอร์ดArduino**



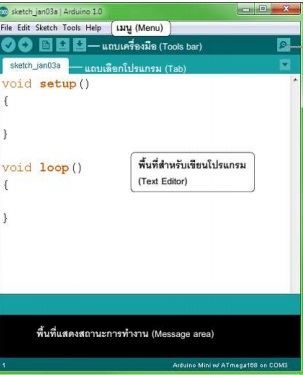
รูปที่ 2.9. บอร์ด Arduino.

Arduino เป็นภาษาอิตาลี ซึ่งใช้เป็นซื่อของโครงการพัฒนาไมโครคอนโทรลเลอร์ตระกูล VARแบบ Open Source ที่ได้รับการปรับปรุงจากโครงการพัฒนา Open Source ของ VAR อีกโครงการหนึ่งที่มีซื่อว่า Wiring เลือกใช้ AVR เบอร์ Atmega128 ซึ่งเป็นไมโครคอนโทรลเลอร์ ที่มีจํานวนหน่วยความจําและ I/O ค่อนข้างมากและที่สําคัญ Atmega128 เป็นชิพทีมีตัวถังแบบ SMD จึงทําให้เป็นอุปสรรคสําหรับผู้เริ่มใช้งานการสร้างบอร์ดและต่อวงจรขึ้นมาใช้งานกันเองและบอร์ดจะมีจะมีขนาดค่อนข้างใหญ่ซึ่งอาจดูว่าเกินความจําเป็นสําหรับผู้เริ่มต้น จึงไม่ค่อยได้รับความนิยมเท่าที่ควร แต่หลังจากที่ทีมงานArduino นํา Source code ของ Wiring มาพัฒนาปรับปรุงใหม่โดยให้สามารถใช้กับไมโครคอนโทรเลอร์ขนาดเล็ก เช่น Mega8 Mega 168 Mega328 ได้จึงทําให้ระบบวงจรของบอร์ดมีขนาดเล็กลงกว่าWringมากและยังใช้อุปกรณ์น้อยชิ้นทําให้ง่ายต่อการต่อวงจรใช้งานกันเอง และยังประหยัดต้นทุนการสร้างบอร์ดไปได้มาก ด้วยเหตุนี้เองที่ทําให้ Arduino ได้รับความนิยมจากผู้ใช้งานทั่วโลกเป็นอย่างมากในระยะเวลา  
อันรวดเร็ว  
 Arduino มีจุดเด่นในเรืองของความง่ายต่อการเรียนรู้และการใช้งาน เนื่องจากมีการออกแบบคําสั่งต่างๆ ขึ้นมาสนับสนุนการใช้งาน ด้วยรูปแบบที่ง่ายไม่ซับช้อน ในตลาดไมโครคอนโทรเลอร์มีตัวเลือกมากเช่น Parallax Basic Stam และอีกหลายเจ้าที่มีคุณสมบัติใกล้เคียงกัน คือทําโปรเจคให้ใช้งานง่ายและเน้นการโปรแกรมไมโครคอนโทรเลอร์เป็นหลัก Arduino ก็เช่นเดียวกันแต่มีข้อแตกต่างที่เห็นได้ ชัดเจนคือ  
 - มีราคาไม่แพง เนืองจากมี Source Code และวงจรแจกฟรี สามารถต่อวงจรขึ้นมาใช้งานเอง  
 - ทํางานได้หลายแพลตฟอร์ม โปรแกรมพัฒนา Arduino ทํางานได้ทั้งบนวินโดว์ OSX และบนลี  
นุกช์ ในขณะที่บอร์ดอื่นทํางานได้เฉพาะบนวินโดว์  
 - ใช้งานง่าย มีโปรแกรมพัฒนาที่ไม่ซับช้อน มีโปรแกรมพัฒนา Arduino ใช้งานง่ายสําหรับ  
มือใหม่ และมีความสามารครบความต้องการของนักพัฒนามืออาชีพตัวบอร์ด Arduino ที่ใช้ในโปรเจคนี้จะกล่าวถึงสถาปัตยกรรมของ AVR ขนาด 8บิต โดยในสถาปัตยกรรมAVR ซึ่งออกแบบโดย ATMEL เมื่อปี1996 เป็นชีพียู แบบ RISC มีสถาปัตยกรรมการต่อหน่วยความจําแบบ Havard ซึ่งแยกหน่วยความจําโปรแกรมและหน่วยความจําข้อมูลออกจากกันโดยเด็ดขาด โดยใช้หน่วยความจําแบบ Flash สําหรับเป็นการความจําโปรแกรม และใช้หน่วยความจําแบบSRAM สําหรับหน่วยความจําข้อมูลและนอกจากนี้ยังมีหน่วยความจําแบบ EEPROM ซึ่งสามารถเก็บข้อมูลเอาไว้ได้โดยไม่จําเป็นต้องมีไฟเลี้ยง  
 สําหรับการเก็บข้อมูลออกจากโปรแกรมอย่างชัดเจน สถาปัตยกรรม AVR และ MCS-51  
จะใช้รูปแบบนี้ในการจัดการหน่วยความจํา ส่วนสถาปัตยกรรมแบบ Von-neumann การตัดสินใจว่าจะเก็บโปรแกรมหรือข้อมูลจะแบ่งเก็บอย่างไรจะทําอย่างอิสระ โดยขึ้นอยู่กับโปรแกรมเมอร์ หรืออาจจะเป็นระบบปฏิบัติการเป็นผู้ดําเนินการให้ลักษณะเด่นของสถาปัตยกรรม AVR คือคําสั่งส่วนใหญ่สามารถทํางานได้เสร็จภายใน 1 Clock cycle ตัวชีพียู AVR ขนาด 8 บิต จะแบ่งประเภทการใช้งานได้ 5 กลุ่ม  
 1. Tiny Avr เป็นชีพียูรุ่นเล็กซึ่งต้องการความเล็กกระทัดรัดของวงจร โดยเหมาะกับระบบควบคุมขนาดเล็กๆ ที่ต้องการหน่วยความจําและวงจรสนับสนุนไม่มากนัก ซีพียูในรุ่นนี้มักมีราคาถูกกว่ากลุ่มอื่นๆ  
 2. MegaAvr จะมีซื่อเรียกอีกอย่างหนึ่งว่า Atmega  
โดยมีวงจรสนับสนุนภายในเพิ่มเติมตลอดจนเพิ่มขนาดของหน่วยความจําให้ใช้งานมากกว่าตระกูล Tinyเหมาะกับงานควบคุมทั่วไป  
 3. Xmega เพิ่มความระเอียดของวงจร A/D จากปกติมีความระเอียด 10 บิตในรุ่นเล็กกว่าเป็น12 บิต และวงจร DMA CONTROLLER ซึ่งช่วยลดภาระของชีพียูในการควบคุมการรับส่งข้อมูลระหว่างอุปกรณ์ I/O กับหน่วยความจํา



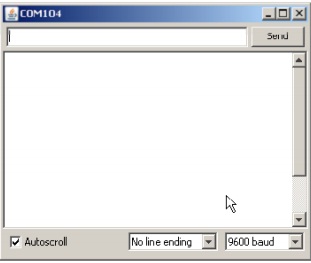
รูปที่ 2.10. ขาต่อใช้งานatmega328

**2.2.1. แนะนํา Arduino 1.0 IDE**  
 ส่วนประกอบของหน้าจอโปรแกรม Arduino 1.0



รูปที่ 2.11. หน้าจอโปรแกรม Arduino 1.0

เมือเรียกให้โปรแกรมทํางาน จะมีหน้าตาดังในรูป2.8 ตัวโปรแกรมประกอบด้วยส่วนต่างๆดังนี้  
 - เมนู (Menu) ใช้เลือกคําสั่งต่างในการใช้งานโปรแกรม  
 - แถบเครื่องมือ ( Toolbar) เป็นการนําคําสั่งที่ใช้งานบ่อยๆ มาสร้างเป็นปุ่มเพื่อเรียกใช้คําสั่งได้รวดเร็วขึ้น  
 - แถบเลือกโปรแกรม (Tabs) เป็นแถบที่ใช้เลือกไฟล์โปรแกรมแต่ละตัว (กรณีที่เขียนโปรแกรมขนาดใหญ่ประกอบด้วยไฟล์หลายตัว)  
 - พื้นที่เขียนโปรแกรม (Text editor) เป็นพื้นที่สําหรับเขียนโปรแกรมภาษา C/C++  
 - พื้นที่แสดงสถานะการทํางาน (Message area) เป็นพื้นที่โปรแกรมใช้แจ้งสถานะการทํางานของโปรแกรมเช่นผลการคอมไพล์โปรแกรม  
 - พื้นที่แสดงข้อมูล (Text area) ใช้แจ้งว่าโปรแกรมที่ผ่านการคอมไพล์แล้วมีขนาดกี่ไบต์  
 - ปุ่มสําหรับเปิดหน้าต่างSerial Monitor ปุ่มนี้จะอยู่ทางมุมบนด้านขวามือคลิกปุ่มนี้เมื่อต้องการเปิดหน้าต่างสื่อสารและแสดงข้อมูลอนุกรมโดยต้องมีการต่อฮาร์ดแวร์ Arduino และเลือกพอร์ตการ

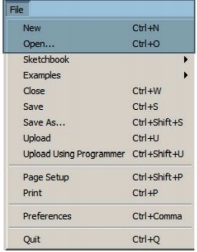


รูปที่ 2.12. หน้าต่าง Serial Monitor

หน้าต่าง Serial Monitor มีบทบาทค่อนข้างมากในการใช้แสดงผลการทําานของโปรแกรมแทนการใช้อุปกรณ์แสดงผลอื่นๆเนื่องจาก Arduino ได้เตรียมคําสั่งสําหรับใช้แสดงค่าของตัวแปรที่ต้องการดูผลการทํางานไว้แล้วนั่นคือSerial.print ส่วนการส่งข้อมูลจากคอมพิวเตอร์ไปยังฮาร์ดแวร์ Arduino หรือแผงวงจรควบคุมให้พิมพ์ข้อความและคลิกปุ่ม Send ในการรับส่งข้อมูลต้องกําหนดอัตราเร็วในการถ่ายทอดข้อมูลหรือบอดเรต(baud rate) ให้กับโปรแกรมในคําสั่งSerial.beginกรณีที่ใช้ งานกับคอมพิวเตอร์Mcintosh หรือคอมพิวเตอร์ที่ติดตั้งระบบปฏิบัติการLinux ตัวฮาร์ดแวร์ของ Arduino จะรีเซ็ตเมื่อเริ่มเปิดใช้งาน Serial monitor

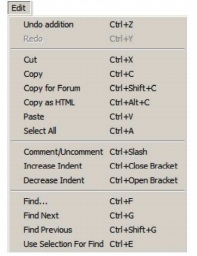
**2.2.2. เมนูบาร์**  
 เป็นส่วนที่แสดงรายการ(เมนู )ของคําสั่งต่างๆของโปรแกรมประกอบด้วย

**2.2.2.1. เมนู File** ใน Arduino จะเรียกโปรแกรมที่พัฒนาขึ้นว่า สเก็ตช์ (Sketch) ในโปรแกรม  
ของผู้ใช้งานอาจมี ไฟล์โปรแกรมหลายตัวจึงเรียกรวมว่าเป็นสเก็ตช์บุ๊ก (Sketchbook) เมนูนี้เกี่ยวข้อง



รูปที่ 2.13. เมนู File

- New ใช้สร้างไฟล์สเก็ตช์ตัวใหม่เพื่อเริ่มเขียนโปรแกรมใหม่  
- Open ใช้เปิดสเก็ตช์ที่บันทึกไว้ก่อนหน้านี้  
- Sketchbook ใช้เปิดไฟล์สเก็ตช์ล่าสุดที่เปิดใช้งานเสมอ  
- Example ใช้ในการเลือกเปิดไฟล์สเก็ตช์ตัวอย่างที่บรรจุและรวบรวมไว้ในโฟลเดอร์ของ  
โปรแกรม Arduino1.0  
- Save ใช้ในการบันทึกไฟล์สเก็ตช์ปัจจุบัน  
- Save as ใช้บันทึกไฟล์สเก็ตช์โดยเปลี่ยนชื่อไฟล์  
- Upload to I/O board ใช้อัปโหลดโปรแกรมไปยังแผงวงจรหรือฮาร์ดแวร์ของ Arduino  
- Page setup ตั้งค่าหน้ากระดาษของไฟล์สเก็ตช์ปัจจุบัน  
- Print สั่งพิมพ์โค้ดของไฟล์สเก็ตช์ปัจจุบันออกทางเครื่องพิมพ์  
- Preference ใช้กําหนดค่าการทํางานของโปรแกรม



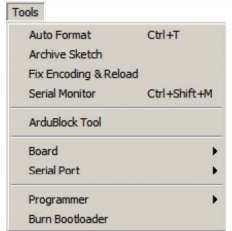
รูปที่ 2.14. เมนู Edit

- Undo ยกเลิกคําสั่งหรือการพิมพ์ครั้งสุดท้าย  
- Redo ทําซํ้าคําสั่งหรือการพิมพ์ครั้งสุดท้าย  
- Cut ตัดข้อความที่เลือกไว้ไปเก็บในคลิปบอร์ดของโปรแกรม  
- Copy คัดลอกข้อความที่เลือกไว้มาเก็บในคลิปบอร์ดPaste นําข้อความที่อยู่ในคลิปบอร์ดมา  
แปะลงในตําแหน่งที่เคอร์เซอร์ชี้อยู่  
- Select All เลือกข้อความทั้งหมด  
- Comment/Uncomment ใช้เติมเครื่องหมาย //เพื่อสร้างหมายเหตุหรือคําอธิบายลงใน  
โปรแกรมหรือยกเลิกหมายเหตุด้วยการนําเครื่องหมาย // ออก  
- Find ค้นหาข้อความ  
- Find Next ค้นหาข้อความถัดไป  
- Find Previous ค้นหาข้อความก่อนหน้า  
- Use Selection For Find ค้นหาด้วยการเลือกคําหรือข้อความ



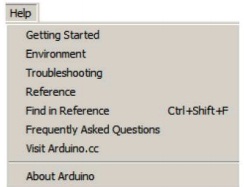
รูปที่ 2.15. เมนูSketch

- Verify/Compile ใช้คอมไพล์แปลโปรแกรมภาษาซีให้เป็นภาษาเครื่อง  
- Show Sketch folder สั่งเปิดโฟลเดอร์ที่เก็บโปรแกรมของผู้ใช้  
- Add file เพิ่มไฟล์ให้กับสเก็ตช์บุ๊กปัจจุบันเมื่อใช้คําสั่งนี้โปรแกรม Arduino จะทําการคัดลอก  
ไฟล์ที่เลือกไว้มาเก็บไว้ในโฟลเดอร์เดียวกันกับโปรแกรมที่กําลังพัฒนา  
- Import Library เป็นคําสั่งเรียกใช้ไลบรารีเพิ่มเติมเมื่อคลิกเลือกคําสั่งนี้แล้วโปรแกรม Arduino  
IDE จะแสดงไลบรารีให้เลือกเมื่อเลือกแล้วโปรแกรมจะแทรกบรรทัดคําสั่ง#include ลงในส่วนต้นของไฟล์

****

รูปที่ 2.16. เมนู Tools

- Auto Format จัดรูปแบบของโค้ดโปรแกรมให้สวยงาม  
- Archive Sketch สั่งบีบอัดไฟล์โปรแกรมทั้งโฟลเดอร์หลักและโฟลเดอร์ย่อยของไฟล์สเก็ตช์  
ปัจจุบันไฟล์ที่สร้างใหม่จะมีชื่อเดียวกับไฟล์สเก็ตช์ปัจจุบันต่อท้ายด้วย -510123.zip  
- Board เลือกฮาร์ดแวร์ของบอร์ดไมโครคอนโทรลเลอร์ Arduino สําหรับเลือกบอร์ด  
- Serial Port เลือกหมายเลขพอร์ตของคอมพิวเตอร์ที่ต่อกับฮาร์ดแวร์ Arduino และบอร์ด  
**2.2.2.5. เมนู Help** เมื่อต้องการความช่วยเหลือหรือข้อมูลเกี่ยวกับโปรแกรมให้เลือกเมนูนี้เมื่อเลือก

****

รูปที่ 2.17. เมนู Help

**2.2.2.6. แถบเครื่องมือ (ทูลบาร์ : Tools bar)** สําหรับคําสั่งที่มีการใช้บ่อยๆตัวโปรแกรม Arduino1.0 จะนํามาสร้างเป็นปุ่ม  
บนแถบเครื่องมือ เพื่อให้สามารถคลิกเลือกได้ทันทีปุ่มต่างๆบนแถบเครื่องมือมีดังนี้  
 - Verfy/Compile ใช้ตรวจสอบการเขียนคําสั่งในโปรแกรมว่ามีถูกต้องตามหลักไวยกรณ์ หรือไม่และคอมไพล์โปรแกรม  
 - Upload to I/O Board ใช้อัปโหลดโปรแกรมที่เขียนขึ้นไปยังบอร์ ดหรือฮาร์ดแวร์Arduino  
ก่อนจะอัปโหลดไฟล์ต้องแน่ใจว่าได้บันทึกไฟล์และคอมไพล์ไฟล์สเก็ตช์เรียบร้อยแล้ว  
 - New ใช้สร้างสเก็ตไฟล์ (ไฟล์โปรแกรม) ตัวใหม่  
 - Open ใช้ แทนเมนู File > Sketchbook เพื่อเปิดสเก็ตช์ (ไฟล์โปรแกรม) ที่มีในเครื่อง  
 - Save ใช้บันทึกไฟล์สเก็ตชบุ๊กทีเขียนขึ้น

**2.3. โครงสร้างโปรแกรมของ Arduino** ในการเขียนโปรแกรมสําหรับบอร์ด จะต้องเขียนโปรแกรมโดยใช้ภาษาของArduino (Arduinoprogramming language) ซึ่งตัวภาษาของ Arduino เองก็นําเอาโอเพ่นซอร์สโปรเจ็กต์ ชื่อ wiring มาพัฒนาต่อภาษาของ Arduino แบ่งได้เป็น 2 ส่วนหลักคือ  
 1. โครงสร้างภาษา (structure) ตัวแปรและค่าคงที่  
 2. ฟังก์ชั่น (function)  
 ภาษาของ Arduino จะอ้างอิงตามภาษา C/C++ จึงอาจกล่าวได้ว่าการเขียนโปรแกรมสหรับArduinoก็ คือการเขียนโปรแกรมภาษา C โดยเรียกใช้ฟังก์ชั่นและไลบรารี ที่ ทาง Arduinoได้เตรียมไว้ให้แล้วซึ่งสะดวกและทําให้ผู้ที่ไม่มีความรู้ด้านไมโครคอนโทรลเลอร์อย่างลึกซึ้งสามารถเขียนโปรแกรมสั่งงานได้โปรแกรมของ Arduino แบ่งได้เป็นสองส่วนคือ  
 1.void setup()  
 2. void loop()  
 โดยฟังก์ชั่นsetup() เมื่อโปรแกรมทํางานจะทําคําสั่งของฟังก์ชั่นนี้เพียงครั้งเดียวใช้ในการกําหนดค่าเริ่มต้นของการทํางานส่วนฟังก์ชั่นloop() เป็นส่วนทํางานโปรแกรมจะทําคําสั่งในฟังชันก์ต่อเนื่องกันตลอดเวลาโดยปกติใช้กําหนดโหมดการทํางานของขาต่างๆกําหนดการสื่อสารแบบอนุกรมส่วนของ loop()เป็นโค้ดโปรแกรมที่ทํางาน เช่นอ่านค่าอินพุตประมวลผลสั่งงานเอาต์พุตโดยส่วนกําหนดค่าเริ่มต้นเช่นตัวแปรจะต้องเขียนที่ส่วนหัวของโปรแกรมก่อนถึงตัวฟังก์ชั่นนอกจากนั้นยังต้องคํานึงถึงตัวพิมพ์เล็ก-ใหญ่ ของตัวแปรและฟังชันก์อื่นๆให้ถูกต้องด้วย  
**2.3.1. คําสั่งควบคุมการทํางาน**

**2.3.1.1. คําสั่ง if** ใช้ทดสอบเพื่อกําหนดเงื่อนไขการทํางานของโปรแกรมเช่นถ้าอินพุตมีค่ามากกว่าค่าที่กําหนดไว้ จะให้ทําอะไรโดยมีรูปแบบการเขียนดังนี้  
 if (someVariable > 50)  
 {  
 // do something here  
 }

ตัวโปรแกรมจะทดสอบว่าถ้าตัวแปร someVariable มีค่ามากกว่า 50 หรือไม่ถ้าใช่ให้ทําอะไรถ้าไม่ใช่ให้ข้ามการทํางานในส่วนนี้การทํางานของคําสั่งนี้จะทดสอบเงือนไขที่เขียนในเครื่องหมายวงเล็บถ้าเงื่อนไขเป็นจริงทําตามคําสั่งที่เขียนในวงเล็บปีกกาถ้าเงื่อนไขเป็นเท็จข้ามการทํางานส่วนนี้ไปส่วนของการทดสอบเงื่อนไขที่เขียนอยู่ภายในวงเล็บจะต้องใช้ตัวกระทําเปรียบเทียบต่างๆดังนี้

x == y (x เท่ากับ y)  
 x != y (x ไม่เท่ากับ y)  
 x < y (x น้อยกว่า y)  
 x > y (x มากกว่า y)  
 x <= y (x น้อยกว่าหรือเท่ากับ y)  
 x >= y (x มากกว่าหรือเท่ากับ y)

**2.3.1.2. if...else** ใช้ ทดสอบเพื่อกําหนดเงื่อนไขการทํางานของโปรแกรมได้มากกว่าคําสั่ง ifธรรมดาโดยสามารถกําหนดได้ว่าถ้าเงื่อนไขเป็นจริงให้ทําอะไรถ้าเป็นเท็จให้ทําอะไรเช่นถ้าค่าอินพุตอะนาลอกที่อ่านได้น้อยกว่า 500 ให้ทําอะไรถ้าค่ามากกว่าหรือเท่ากับ 500 ให้ทําอีกอย่างจะเขียนคําสั่งได้ดังนี้  
 if (pinFiveInput < 500)  
 {  
 // do Thing A  
 }  
 else  
 {  
 // do Thing B  
 }  
 หลังคําสั่ง else สามารถตามด้วยคําสั่ง if สําหรับการทดสอบอื่นๆทําให้รูปแบบคําสั่งกลายเป็น if....else...if เป็นการทดสอบเงื่อนไขต่างๆ เมื่อเป็นจริงให้ทําตามที่ต้องการ  
 **2.3.1.3. คําสั่ง for()** คําสั่งนี้ใช้เพื่อสั่งให้คําสั่งที่อยู่ภายในวงเล็บปีกกาหลัง for มีการทํางานซํ้ากันตามจํานวนรอบที่ต้องการคั่งนี้มีปีระโยชนมากสําหรับการทํางานใดๆที่ต้องการทําซ้ํากันและทราบจํานวนรอบของการทํา ช้ําที่แน่นอนมักใช้คู่กับตัวแปรอะเรย์ในการเก็บสะสมาค่าที่อ่านได้จากอินพุตอะนาลอกหลายๆขาที่มี่ขาต่อเนื่องกัน  
 for (initialization; condition; increment)  
 {  
 //statement(s);  
 }  
 เริ่มต้นด้วย initialization ใช้กําหนดค่าเริ่มต้นของตัวแปรควบคุมการวนรอบในการทํางานแต่ละรอบจะทดสอบ condition ถ้าเงื่อนไขเป็นจริงทําตามคําสั่งในวงเล็บปีกกาแล้วมาเพิ่มหรือลดค่าตัวแปร ตามที่สั่งในincrement แล้วทดสอบเงื่อนไขอีกทําซํ้าจนกว่าเงื่อนไขเป็นเท็จ  
**2.3.1.4. switch-case**  
 ใช้ทดสอบเงื่อนไขเพื่อกําหนดการทํางานของโปรแกรมถ้าตัวแปรที่ทดสอบตรง  
กับเงื่อนไขใดก็ให้ทํางานตามที่กําหนดไว้พารามิเตอร์  
 - Var ตัวแปรที่ต้องการทดสอบว่าตรงกับเงื่อนไขใด  
 - default ถ้าไม่ตรงกับเงื่อนไขใดๆ เลยให้ทําคําสั่งต่อท้ายนี้  
 - break เป็นส่วนสําคัญมากใช้เขียนต่อท้าย case ต่างๆเมื่อพบเงื่อนไขนั้นแล้วทําตามคําสั่งต่างๆ  
  
 แล้วให้หยุดการทํางานของคําสั่ง switch-case ถ้าลืมเขียน break เมื่อพบเงื่อนไขทําตามเงื่อนไขแล้วโปรแกรมจะทํางานตามเงื่อนไขต่อไปเรื่อยๆ จนกว่าจะพบคําสั่ง break   
 switch (var)  
 {  
 case 1:  
 //do something when var == 1  
 break;  
 case 2:  
 //do something when var == 2  
 break;  
 default:  
 // if nothing else matches, do the default 2.3.1.5. คําสั่ง while  
 เป็นคําสั่งวนรอบโดยจะทําคําสั่งที่เขียนในวงเล็บปีกกาอย่างต่อเนื่องจนกว่า  
เงื่อนที่เขียนในวงเล็บของคําสั่ง while() จะเป็นเท็จคําสั่งที่ให้ทําซํ้าจะต้องมีการเปลี่ยนแปลงค่าตัวแปรที่ใช้ทดสอบเช่นมีการเพิ่มค่าตัวแปรหรือมีเงื่อนไขภายนอกเช่นอ่านค่าจากตัวตรวจจับได้เรียบร้อยแล้วให้หยุดการอ่านค่ามิฉะนั้นเงื่อนไขในวงเล็บของ while() เป็นจริงตลอดเวลาทําให้คําสั่ง while ทํางานวนรอบไปเรื่อยๆไม่รู้จบรูปแบบคําสั่ง

while(expression)  
 {// statement(s)  
 }

ตัวอย่าง  
 var = 0;  
 while(var < 200)  
 {  
 // do something repetitive 200 times  
 var++;  
 }

**2.3.2. ตัวกระทําทางคณิตศาสตร์**  
 ประกอบด้วยตัวกระทํา5 ตัวคือ + (บวก) - (ลบ) \* (คู ณ) / (หาร) และ % (หารเอาเศษ)  
  
 **2.3.2.1. ตัวกระทําทางคณิตศาสตร์ บวก ลบ คู ณ และหาร** ใช้หาค่าผลรวมผลต่างผลคูณและผลหารค่าของตัวถูกกระทําสองตัวโดยให้คําตอบมีประเภทตรงกับตัวถูกกระทําทั้งสองตัว เช่น 9/4 ให้คําตอบเท่ากับ 2 เนื่องจากทั้ง 9 และ 4 เป็น  
ตัวแปรเลขจํานวนเต็ม (int)นอกจากนี้ตัวกระทําทางคณิตศาสตร์อาจทําให้เกิดโอเวอร์โฟลว (overflow)ถ้าผลลัพธ์ที่ได้มีขนาดใหญ่เกินกว่าจะสามารถเก็บในตัวแปรประเภทนั้นถ้าตัวที่ถูกกระทําต่างประเภทกันผลลัพธ์ได้เป็นจะมี ขนาดใหญ่ขึ้นเท่ากับประเภทของตัวแปรที่ใหญ่ที่สุด เช่น 9/4 = 2 หรือ 9/4.0 =2.25)  
 รูปแบบคําสั่ง  
 - result = value1 + value2;  
 - result = value1 - value2;  
 - result = value1 \* value2;  
 - result = value1 / value2;  
 พารามิเตอร์  
 - value1 เป็นค่าของตัวแปรหรือค่าคงที่ใดๆ  
 - value2 เป็นค่าของตัวแปรหรือค่าคงที่ใดๆ  
**2.3.2.2. ตัวกระทํา % หารเอาเศษ**  
 ใช้หาค่าเศษที่ ได้ของการหารเลขจํานวนเต็ม 2 ตัวตัวกระทําหารเอาเศษไม่สามารถใช้งานกับตัวแปรเลขทศนิยม (float)  
 รูปแบบคําสั่ง  
 result = value1 % value2;  
 พารามิเตอร์  
 value1 - เป็ นตั วแปรประเภท byte,char,int หรื อ long  
 value2 - เป็ นตั วแปรประเภท byte,char,int หรื อ long  
 ผลที่ ได้เศษจากการหารค่าเลขจํานวนเต็มเป็นข้อมูลชนิดเลขจํานวนเต็ม

ตัวอย่าง  
 x = 7 % 5; // x now contains 2  
 x = 9 % 5; // x now contains 4  
 x = 5 % 5; // x now contains 0  
 x = 4 % 5; // x now contains 4

**2.3.3. ตัวกระทําเปรียบเทียบ** ใช้ประกอบกับคําสั่ง if() และ while() เพื่อทดสอบเงื่อนไขหรือเปรียบเทียบค่าตัวแปรต่างโดยจะเขียนเป็นนิพจน์อยู่ภายใเครื่องหมาย ()  
 x == y (x เท่ากับ y)  
 x != y (x ไม่เท่ากับ y)  
 x < y (x น้อยกว่า y)  
 x > y (x มากกว่า y)  
 x <= y (x น้อยกว่าหรือเท่ากับ y)  
**2.3.4. ตัวกระทําทางตระกะ**  
 ใช้ในการเปรียบเทียบของคํา สั่ง if() มี 3 ตัวคือ &&, || และ ! 2.3.4.1. && (ตรรกะ และ)  
 ให้ค่าเป็นจริงเมื่อผลการเปรียบเทียบทั้งสองข้างเป็นจริงทั้งคู่  
 ตัวอย่าง  
 if (x > 0 && x < 5)  
 {  
 // ...  
 }  
 ให้ค่าเป็นจริงเมื่อ x มากกว่า 0 และน้อยกว่า 5 (มี ค่า 1 ถึ4)  
**2.3.4.2. || (ตรรกะ หรือ)** ให้ค่าเป็นจริงเมื่อผลการเปรียบเทียบพบว่ามีตัวแปรใดเป็นจริงหรือเป็นจริงทั้งคู่  
 ตัวอย่าง  
 if (x > 0 || y > 0)  
 {  
 // ...  
 }

ให้ผลเป็นจริงเมื่อ x หรื อ y มี ค่ามากกว่า 0

**2.3.4.3. ! (ใช้กลับผลเป็นตรงกันข้าม)** ให้ค่าเป็นจริงเมื่อผลการเปรียบเทียบเป็นเท็จ  
 ตัวอย่าง if (!x)  
 {  
 // ...  
 }

ให้ผลเป็นจริงถ้า x เป็นเท็จ (เช่น ถ้า x = 0 ให้ผลเป็นจริง)

**2.3.5. ตัวกระทําระดับบิต**   
 ตัวกระทําระดับจะนําบิตของตัวแปรมาประมวลผลใช้ประโยชน์ในการแก้ปัญหาด้านการเขียนโปรแกรมได้หลากหลายตัวกระทําระดับของภาษา C (ซึ่งรวมถึง Arduino) มี 6 ตัวได้แก่ & (bitwiseAND),| (OR), ^ (Exclusive OR), ~ (NOT), << (เลื่อนบิตไปทางขวา) และ >> (เลื่อนบิตไปทางซ้าย)

**2.3.5.1. ตัวกระทําระดับบิต AND (&)**  
 คําสั่ง AND ในระดับบิตของภาษา C เขียนได้โดยใช้ & หนึ่งตัวโดยต้องเขียนระหว่างนิ พจน์ หรือตัวแปรที่ เป็นเลขจํานวนเต็มการทํางานจะนําข้อมูลแต่ละบิตของตัวแปรทั้งสองตัวมากระทําทางตรรกะและ(AND) ถ้าอินพุตทั้งสองตัวเป็น “1” ทั้งคู่เอาต์พุตเป็น “1” กรณี อื่นๆ เอาต์พุตเป็น“0” ในการดูให้คู่ของตัวกระทําตามแนวตั้ง

ตัวอย่าง  
 int a = 92; // in binary: 0000000001011100  
 int b = 101; // in binary: 0000000001100101   
 Int c = a & b; // result: 0000000001000100  
 // or 68 in decimal.  
 ในตัวอย่างนี้จะนําข้อมูลทั้ง 16 บิตของตัวแปร a และ b มากระทํา ทางตรรกะ AND แล้วนําผลลัพธ์ที่ได้ทั้ง 16 บิตไปเก็บที่ตัวแปร c ซึ่งได้ ค่าเป็น 01000100 ในเลขฐานสองหรือเท่ากับ 68 ฐานสิบนิยมใช้ตัวกระทําระดับบิต AND เพื่อเลือกข้อมูลบิตที่ต้องการ (อาจเป็นหนึ่งบิตหรือหลายบิต) จากตัวแปรint ซึ่งการเลือกเพียงบางบิตนี้จะเรียกว่า masking  
**2.3.5.2. ตัวกระทําระดับบิต OR ( | )** คําสั่งระดับบิต OR ของภาษาซีเขียนได้โดยใช้เครื่องหมาย |หนึ่งตัวโดยต้องเขียนระหว่างนิพจน์ หรือตัวแปรที่เป็นเลขจํานวนเต็ม การทํางานะนําข้อมูลแต่ละบิตของตัวแปรทั้งสองตัวมากระทําทางตรรกะ หรือ (OR) โดยมีกฎดังนี้อินพุตตัวใดตัวหนึ่งหรือทั้งสองตัวเป็น “1” เอาต์พุตเป็น “1” กรณีที่อินพุตเป็น “0” ทั้งคู่ เอาต์พุตจึงจะเป็น “0”  
ตัวอย่าง  
 ส่วนของโปรแกรมแสดงการใช้ตัวกระทํา ระดับบิต OR  
 int a = 92; // in binary: 0000000001011100  
 Int b = 101; // in binary: 0000000001100101  
 int c = a | b; // result: 0000000001111101  
 // or 125 in decimal  
**2.3.5.3. คําสั่งระดับบิต Exclusive OR (^)**  
 เป็นโอเปอร์เตอร์พิเศษที่ไม่ค่อยได้ใช้ในภาษา C/C++ ตัวกระทําระดับบิตexclusive OR (หรือ XOR)จะเขียนโดยใช้สัญลักษณ์เครื่องหมาย ^ ตัวกระทํานี้มีการทํางานใกล้เคียงกับตัวกระทําระดับบิต OR แต่ต่างกันเมื่ออินพุตเป็น “1” ทั้งคู่จะให้เอาต์พุตเป็น “0”หรือกล่าวได้อีอย่างว่าตัวกระทําระดับบิต XOR จะให้เอาต์พุตเป็น “0” เมื่ออินพุตทั้งสองตัวมีค่าเหมือนกันและให้เอาต์พุตเป็น“1” เมื่ออินพุตทั้งสองมีค่าต่างกัน  
 ตัวอย่าง  
 int x = 12; // binary: 1100  
 int y = 10; // binary: 1010  
 int z = x ^ y; // binary: 0110, or decimal 6  
 ตัวกระทําระดับบิต XOR จะใช้มากในการสลับค่าบางบิตของตัวตัวแปร int เช่น กลับจาก “0”เป็น“1” หรื อกลับจาก “1” เป็น “0”เมื่อใช้ตัวกระทําระดับบิต XOR ถ้าบิตของ mask เป็น “1” ทําให้บิตนั้นถูกสลับค่า ถ้า mask มีค่าเป็น“1” บิตนั้นมีค่าคงเดิมตัวอย่างต่อไปนี้เป็นโปรแกรมแสดงการสั่งให้ขาดิจิตอล 5 (Di 5) มี การกลับลอจิกตลอดเวลา  
**2.3.5.4. ตัวกระทําระดับบิต NOT (~)** ตัวกระทําระดับบิต NOT จะเขียนโดยใช้สัญลักษณ์เครื่องหมายตัวกระทํานี้จะใช้งานกับตัวถูกกระทําเพียงตัวเดียวที่อยู่ขวามือโดยทําการสลับบิตทุกบิตให้มีค่าตรงกันข้ามคือ จาก “0”เป็น “1” และจาก “1” เป็น“0”  
 0 1 Operand1  
 1 0 ~ operand1  
 int a = 103; // binary: 0000000001100111  
 int b = ~a; // binary: 1111111110011000  
 เมื่อกระทําแล้วทําให้ตัวแปร b มี ค่า -104 (ฐานสิบ) ซึ่งคําตอบที่ได้ติดลบเนื่องจากบิตที่มีความสําคัญสูงสุด (บิตซ้ายมือสุด) ของตัวแปร int อันเป็นบิตแจ้งว่าตัวเลขเป็นบวกหรือลบมีค่าเป็น“1”แสดงว่าค่าที่ได้นี้ติดลบโดยในคอมพิวเตอร์จะเก็บค่าตัวเลขทั้งบวกและลบตามระบบทูคอมพลีเมนต์(2’s complement)

**2.3.5.5. คําสั่งเลื่อนบิตไปทางซ้าย (<<) และเลื่อนบิตไปทางขวา (>>)** ในภาษา C/C++ มี ตัวกระทําเลื่อนบิตไปทางซ้าย << และเลื่อนบิตไปทางขวา>> ตัวกระทํานี้ จะสั่งเลื่อนบิตของตัวถูกกระทําที่เขียนด้านซ้ายมือไปทางซ้ายหรือไปทางขวาตามจํานวนบิตที่ระบุไว้ในด้านขวามือของตัวกระทํา  
รูปแบบคําสั่ง  
 variable << number\_of\_bits  
 variable >> number\_of\_bits  
 พารามิเตอร์  
 variable เป็นตัวแปรเลขจํานวนเต็มที่มีจํานวนบิตน้อยกว่าหรือเท่ากับ 32 บิ ต (หรือตัวแปรประเภทbyte, int หรื อ long)  
ตัวอย่าง  
 int a = 5; // binary: 0000000000000101  
 int b = a << 3; // binary: 0000000000101000,  
 // or 40 in decimal  
 int c = b >> 3; // binary: 0000000000000101,  
 // or back to 5

**2.3.6. ตัวแปร**  
 ตัวแปรเป็นตัวอักษรหลายตัวๆ ที่กําหนดขึ้นในโปรแกรมเพื่อใช้ในการเก็บค่าข้อมูลต่างๆเช่น ค่าที่อ่านได้จากตัวตรวจจับที่ต่ออยู่กับขาพอร์ตอะนาลอกของ Arduino ตัวแปรมีหลายประเภทดังนี้  
**2.3.6.1. char ตัวแปรประเภทตัวอักขระ** เป็นตัวแปรที่มีขนาด 1 ไบต์ (8 บิ ต) มีไว้เพื่อเก็บค่าตัวอักษรตัวอักษรในภาษาซีจะเขียนอยู่ในเครื่องหมายคําพูดขีดเดียวเช่น ‘A’ (สําหรับข้อความที่ประกอบจากตัวอักษรหลายตัวเขียนต่อกันจะเขียนอยู่ในเครื่องหมายคํา พู ดปกติ เช่ น “ABC”) คุ ณสามารถสั่ งกระทํา ทางคณิ ตศาสตร์ กั บตั วอั กษรได้ ในกรณีจะนําค่ารหัสASCII ของตัวอักษรมาใช้ เช่น ‘A’ +1 มี ค่าเท่ากับ 66เนื่องจากค่ารหั ส ASCII ของตัวอักษร A เท่ากับ 65

รูปแบบคําสั่ง  
 char sign = ' ';

พารามิเตอร์  
 char var = ‘x’;  
 var คือชื่อของตัวแปรประเภท char ที่ต้องการ

x คือค่าที่ต้องการกําหนดให้กับตัวแปร ในที่นี้เป็นตัวอักษร หนึ่งตัว  
 **2.3.6.2. byte ตัวแปรประเภทตัวเลข 8บิตหรือ 1 ไบต์**  
ตัวแปร byte ใช้เก็บค่าตัวเลขขนาด 8 บิต มีค่าได้ จาก 0 - 255 ตัวอย่าง  
 byte b = B10010111; // “B” is the binary formatter (151 decimal)  
**2.3.6.3. int ตัวแปรประเภทตัวเลขจํานวนเต็ม**  
 ย่อจาก interger ซึ่งแปลว่าเลขจํานวนเต็ม int เป็นตัวแปรพื้นฐานสําหรับเก็บตัวเลขตัวแปรหนึ่งตัวมีนาด 2 ไบต์ เก็บค่าได้ จาก -32,768 ถึ ง 32,767 (ค่าตํ่าสุดจาก -215 ค่าสูงสุดจาก(215- 1)ในการเก็บค่าตัวเลขติดลบจะใช้เทคนิคที่เรียกว่าทูคอมพลีเมนต์ (2’s complement) บิตสูงสุดบางทีจะเรียกว่าเป็นบิตเครื่องหมายหรือ sign bit ถ้ามีค่าเป็น “1” แสดงว่าค่าติดลบใน Arduino จะจัดการกับตัวเลขค่าติดลบให้เองทําให้นําค่าตัวแปรไปคํานวณได้อย่างถูกต้องอย่างไรก็ตามเมื่อนําตัวเลขค่าติดลบนี้ไปเลื่อนบิตไปทางขวา (>>) จะมีปัญหาเรื่องค่าของตัวเลขที่ผิดพลาด  
ตัวอย่าง  
 int x  
 x = -32,768;  
 x = x - 1; // x now contains 32,767  
 // - rolls over in neg. direction  
 x = 32,767;  
 x = x + 1; // x now contains -32,768 - rolls over  
**2.3.6.4. unsigned int ตัวแปรประเภทเลขจํานวนเต็มไม่คิดเครื่องหมาย**

ตัวแปรประเภทนี้จะคลายกับตัวแปร int ตรงที่ใช้หน่วยความจํา2 ไบต์ แต่จะเก็บเลขจํานวนเต็ม บวกเท่านั้นโดยเก็บค่า 0 ถึง 65,535 (216 -1)  
รูปแบบคําสั่ง  
unsigned int var = val;พารามิเตอร์

var คือชื่อของตัวแปร int ที่ ต้องการ  
 val คือค่าที่ต้องการกําหนดให้กับตัวแปร  
 ตัวอย่าง  
 unsigned int ledPin = 31;  
 **2.3.6.5. long ตัวแปรประเภทเลขจํานวนเต็ม 32 บิต**

เป็นตัวแปรเก็บค่าเลขจํานวนเต็มที่ขยายความจุเพิ่มจากตัวแปร int โดยตัวแปร long หนึ่งตัวกินพื้นที่หน่วยความจํา 32 บิ ต (4 ไบต์ ) เก็บค่าได้จาก -2,147,483,648 ถึง2,147,483,647

รูปแบบคําสั่ง  
long var = val;  
พารามิเตอร์  
var คือชื่อของตัวแปร long ที่ต้องการ  
val คือค่าที่ต้องการกําหนดให้กับตัวแปร  
ตัวอย่าง  
long time;  
void setup()  
{  
Serial.begin(9600);  
}  
void loop()  
{  
Serial.print("Time: ");  
time = millis();  
Serial.println(time); //prints time since program started  
delay(1000); // wait a second so as not to send massive amounts of data  
}  
2