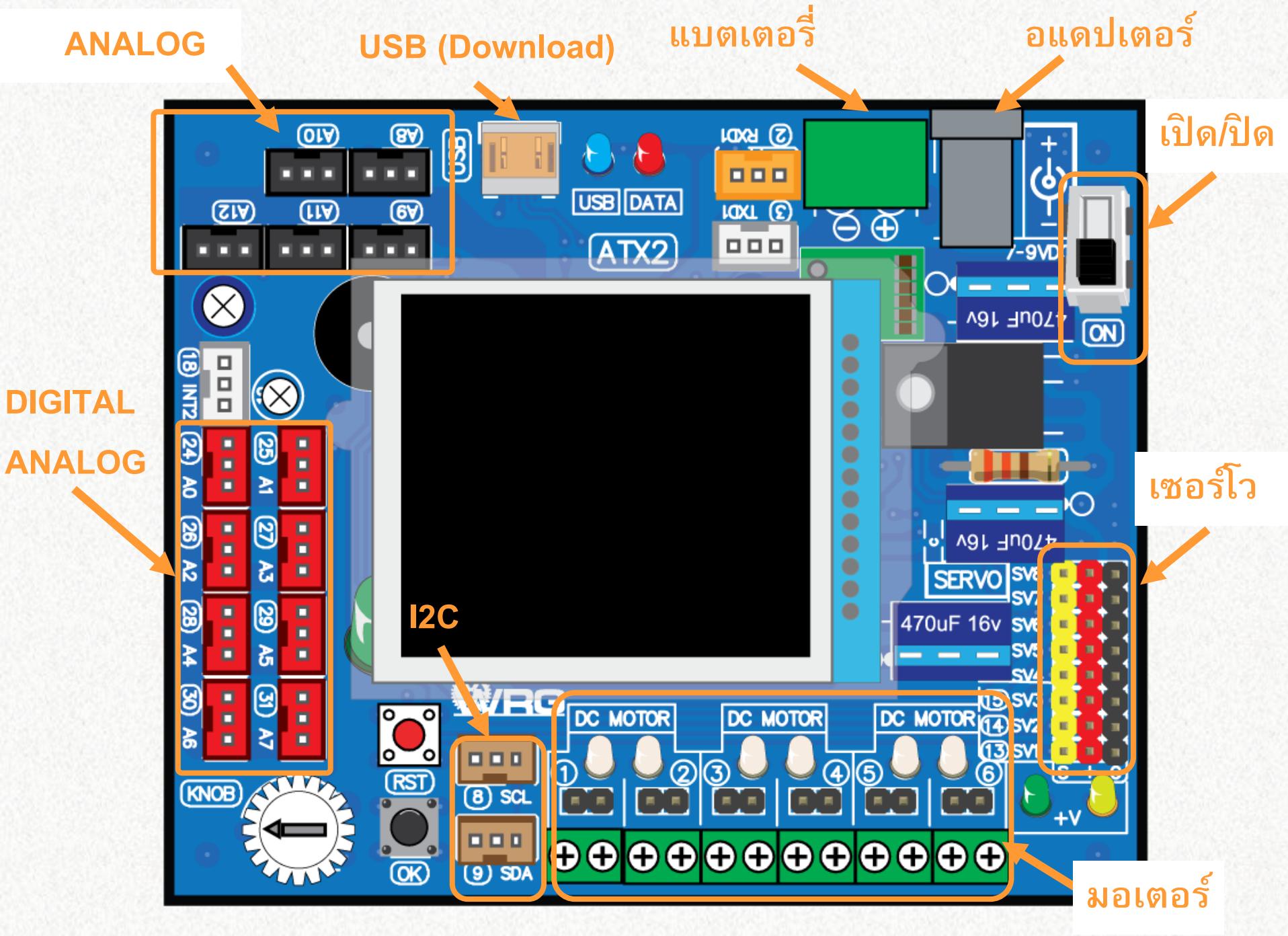


การใช้งานหุ่นยนต์

RQ-BOT



หลักการของระบบควบคุม



สื่อสารอนุกรม 1

ลำโพงเปียโน

ไฟเตือนแบต

knob

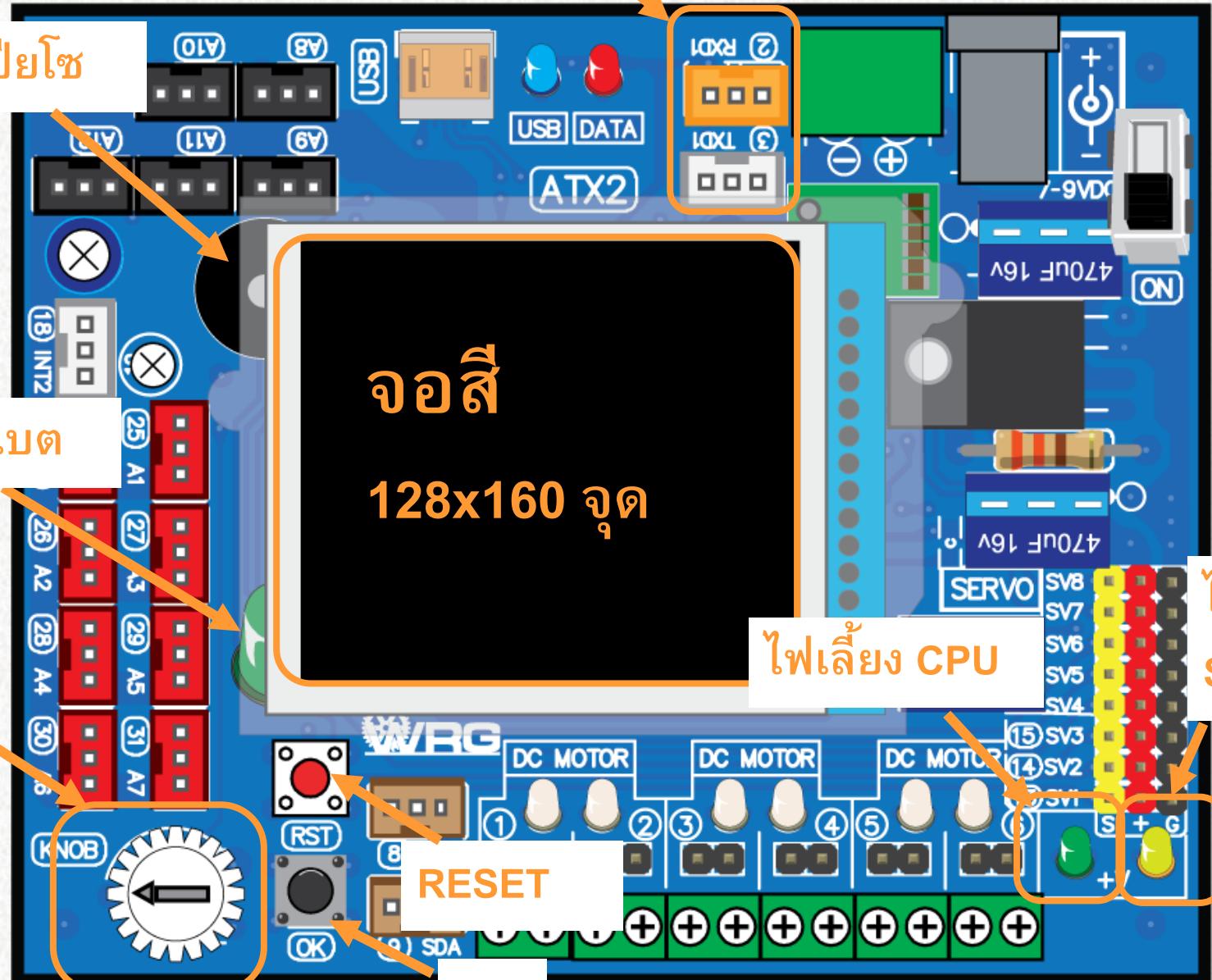
จอสี
128x160 จุด

ไฟเลี้ยง CPU

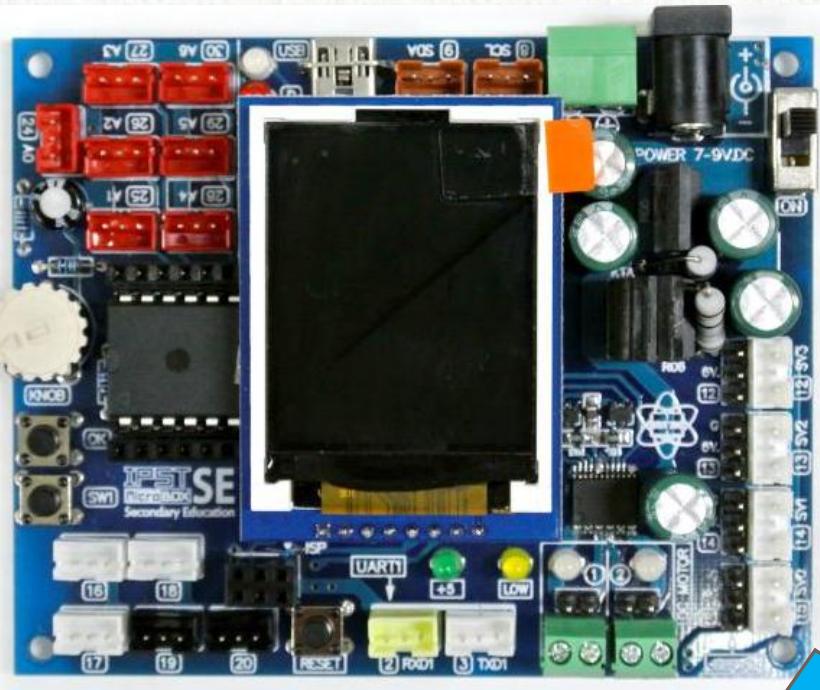
ไฟเลี้ยง
SERVO

RESET

OK



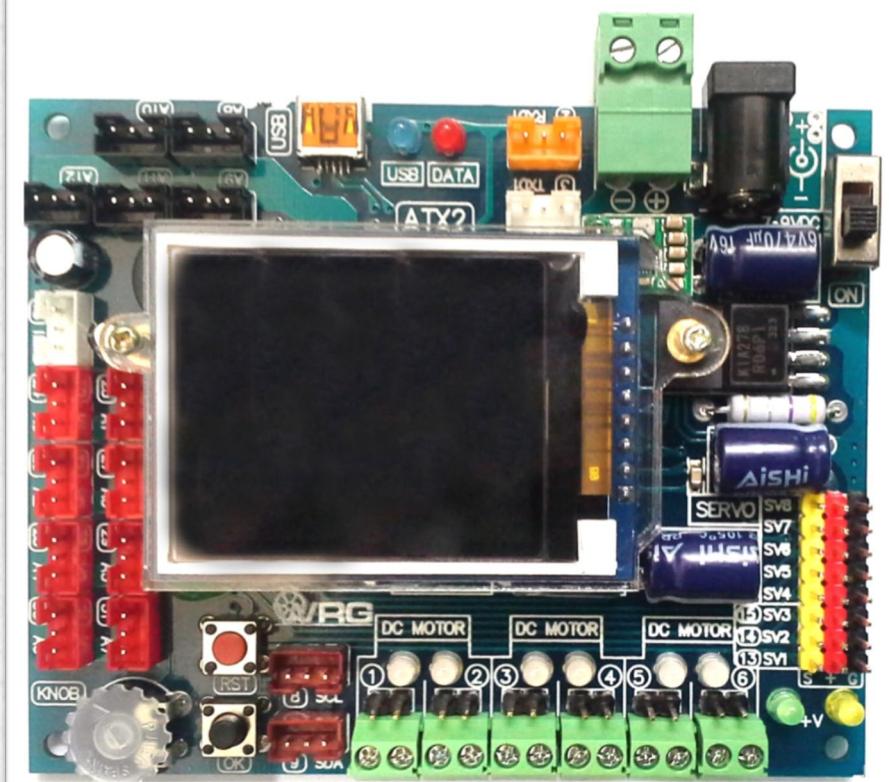
เปรียบเทียบ IPST-SE vs ATX2



IPST-SE

Motor x 6
Servo x 8
I/O x 18

Motor x 2
Servo x 4
I/O x 20



ATX2

คอมพิลे�อร์ Open Source

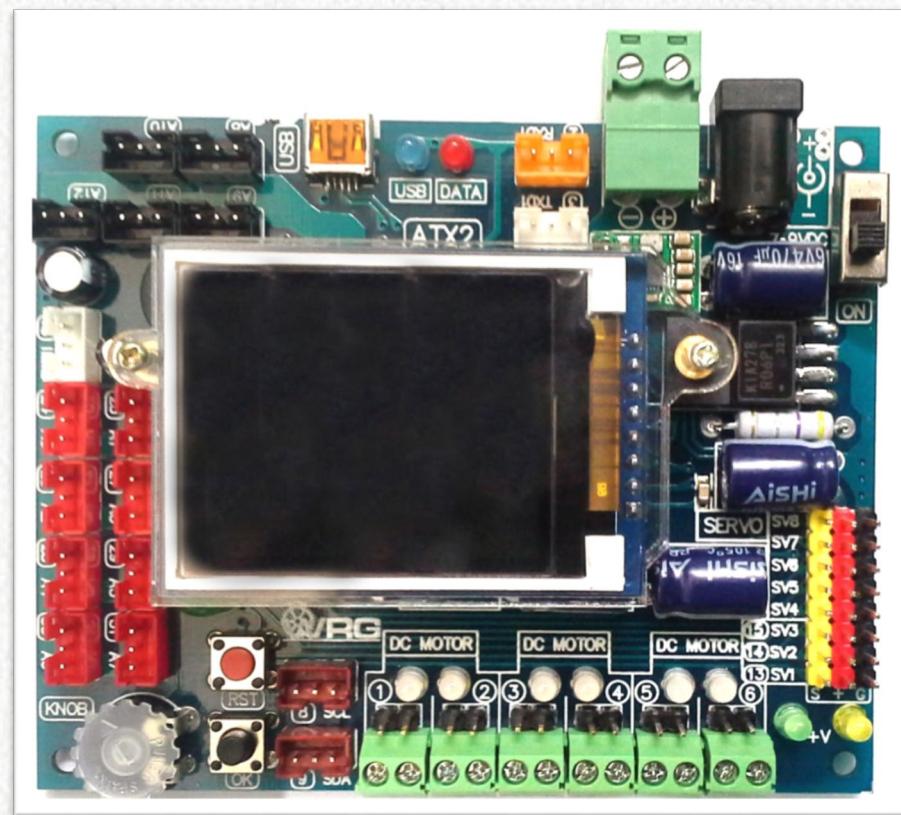
Software

The screenshot shows the Arduino IDE interface with the title bar "ObjectHitterBOT | Arduino 0107". The menu bar includes File, Edit, Sketch, Tools, Help. Below the menu is a toolbar with icons for upload, download, and other functions. The main area displays the C++ code for the "ObjectHitterBOT" sketch. The code includes setup() and loop() functions. The setup() function initializes the servo motor and displays messages on a liquid crystal display (LCD). The loop() function performs a sequence of movements: stopping the motor, driving it to 180 degrees, sleeping, driving it back to 0 degrees, moving backward, sleeping again, and finally turning 90 degrees right. A message at the bottom of the code area says "Done compiling." The status bar at the bottom shows "Binary sketch size: 14,548 bytes (of a 64,512 byte maximum)" and "ATX2, ATmega644P @ 20 MHz on COM3".

```
ObjectHitterBOT | Arduino 0107
File Edit Sketch Tools Help
ObjectHitterBOT §
sleep(50);
ao(); // Stop motor
servo(1,180); // Drive servo motor to 180-degree posit
sleep(1000);
servo(1,0); // Drive servo motor back to 0-degree po
bk(60); // Move backward
sleep(200);
R90(); // 90-degree turn right
}
void setup()
{
  setTextSize(2); // Set text size 2x
  glcd(1,1,"Press OK"); // Show the start message
  glcd(2,1,"to Start");
  sw_OK_press(); // Wait for OK switch pressed
  glcdClear();
  glcd(1,1,"Let's go!"); // Show operation message
}
void loop()
< >
Done compiling.

Binary sketch size: 14,548 bytes (of a 64,512 byte maximum)
53 ATX2, ATmega644P @ 20 MHz on COM3
```

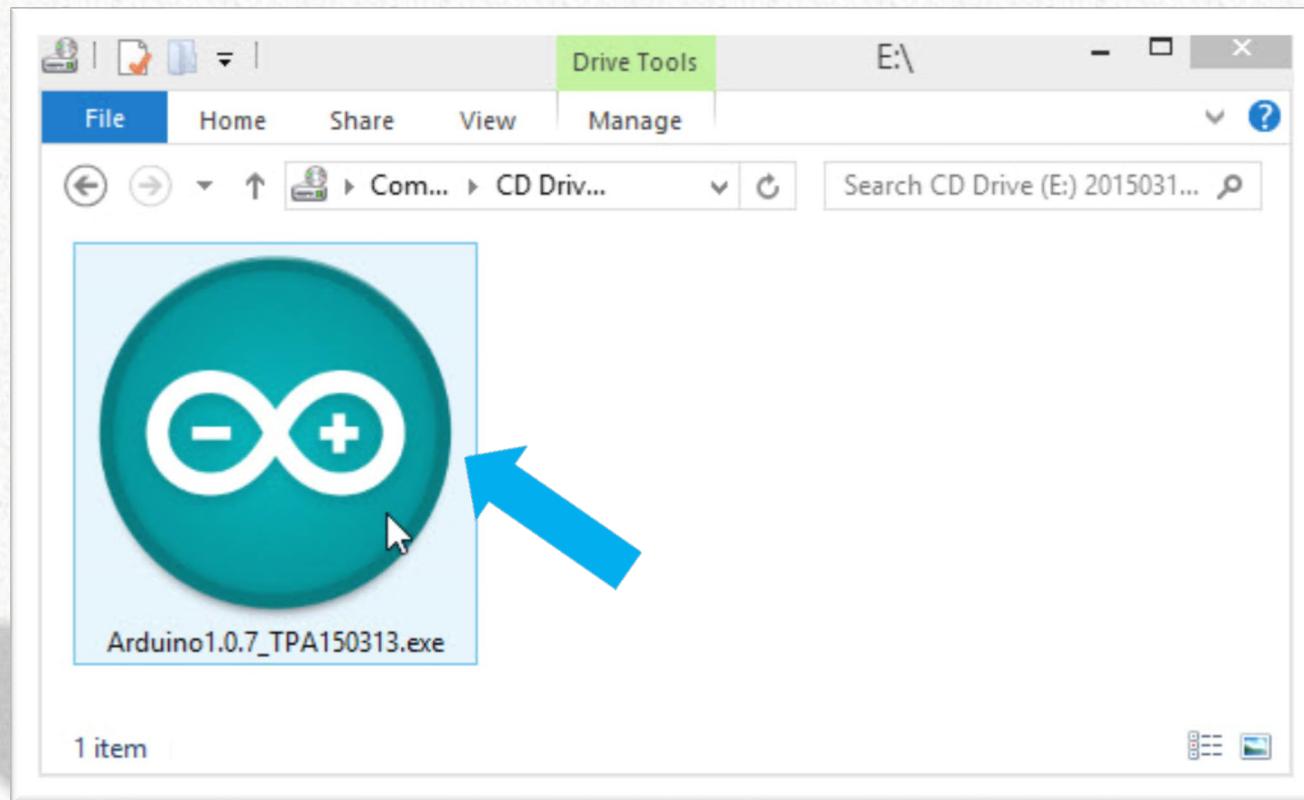
Edit+Compile+Download



Arduino

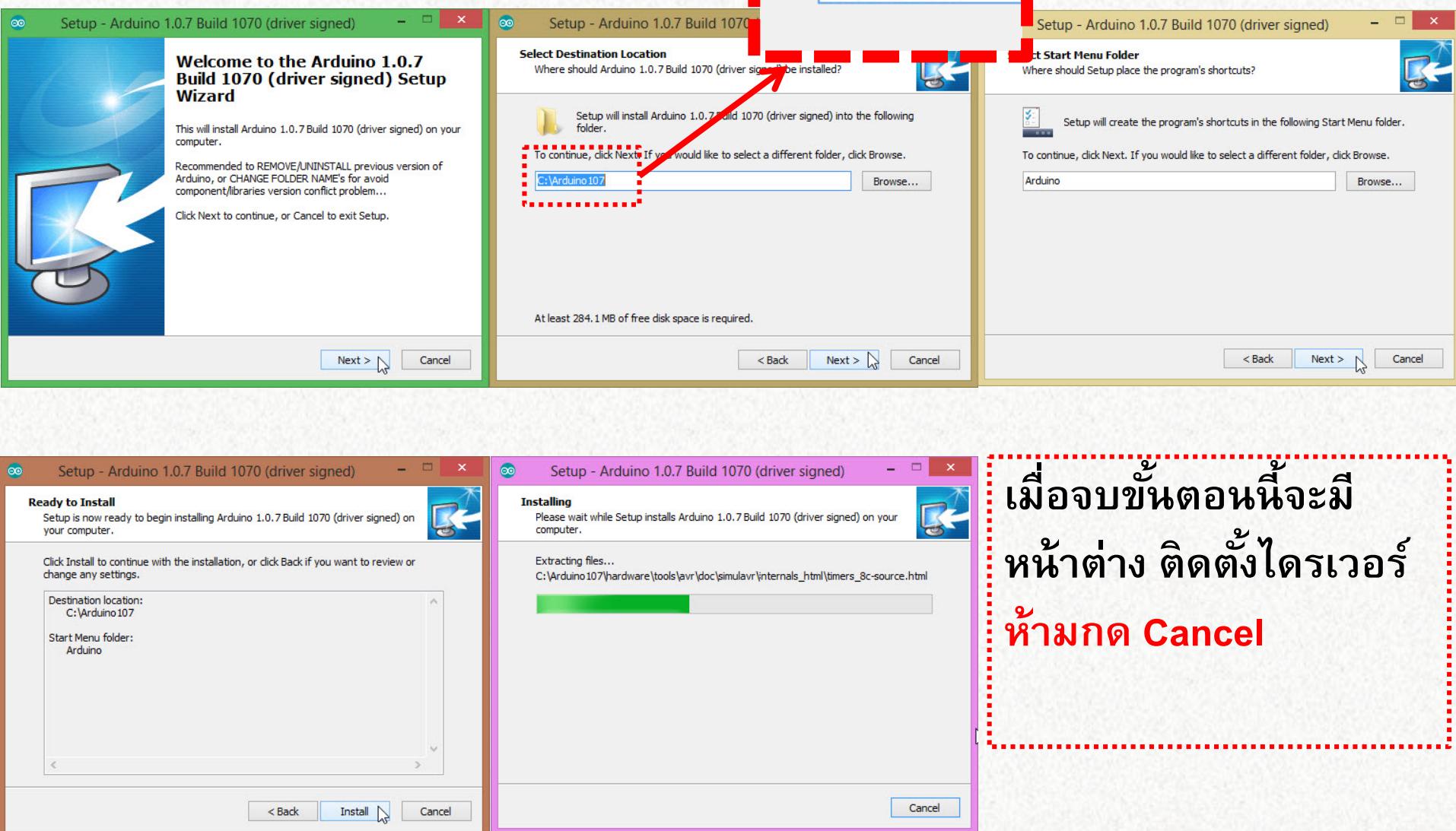
ATX2

ติดตั้งโปรแกรม Arduino 1.07

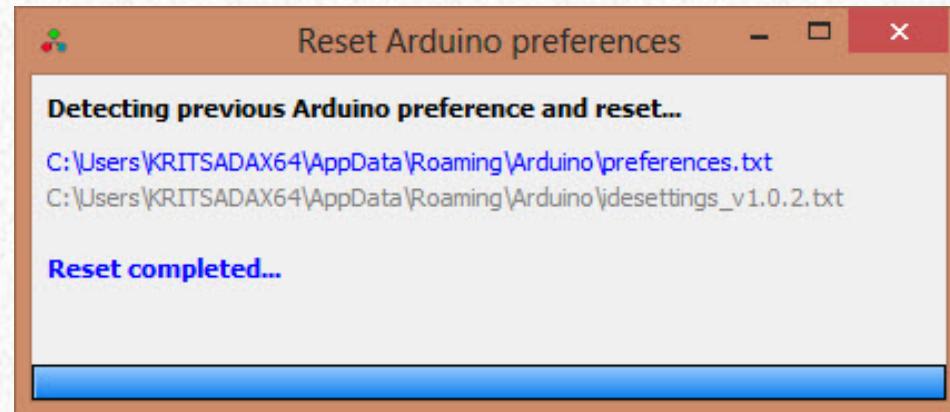
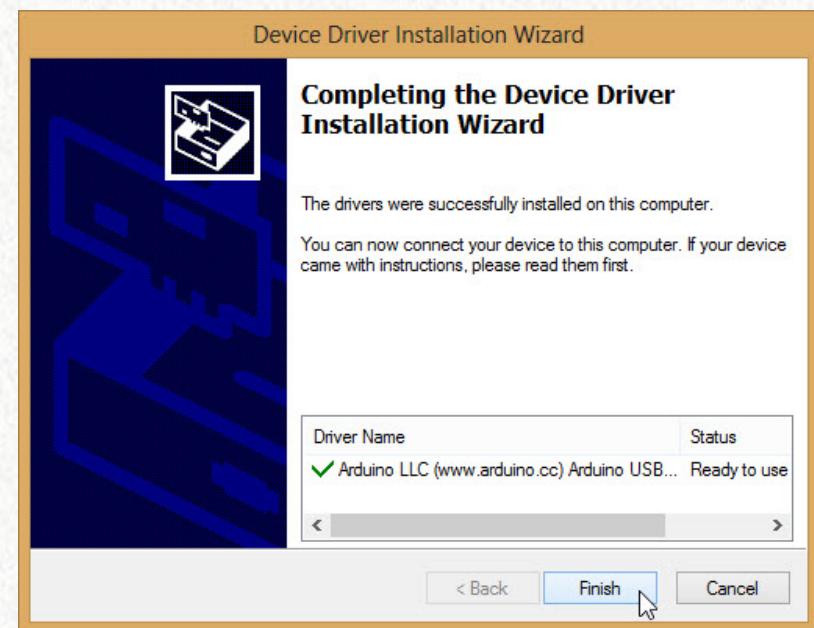
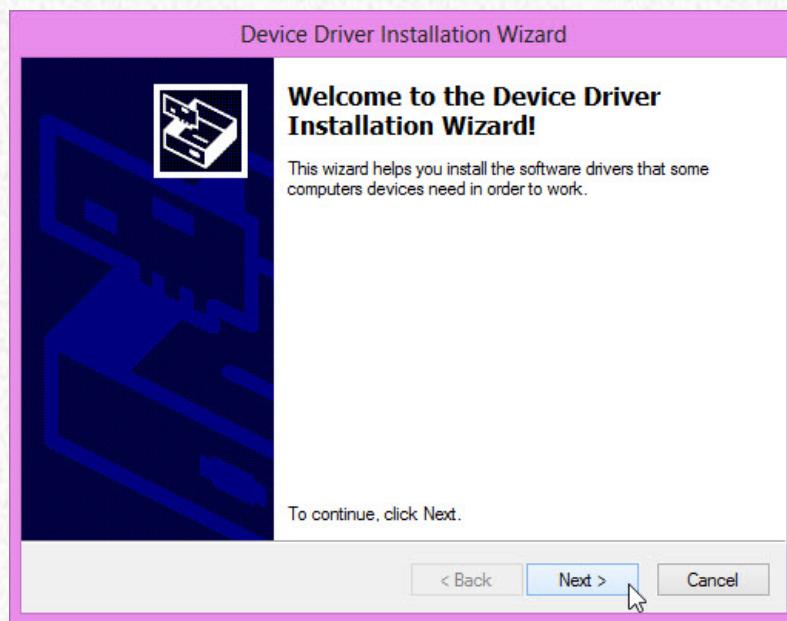


1. เปิดโปรแกรมในแผ่น CDROM
2. ดับเบิลคลิกไฟล์ Arduino 1.07_TPA150313
3. ทำตามขั้นตอนติดตั้งโปรแกรม

ขั้นตอนติดตั้งโปรแกรม

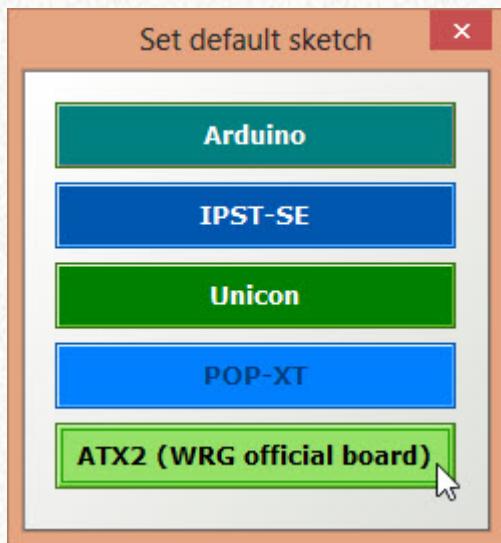


ขั้นตอนติดตั้งไดรเวอร์



เพื่อให้คอมพิวเตอร์รู้จักกับบอร์ด ATX2

ขั้นตอนติดตั้งไดรเวอร์



```
sketch_mar14a | Arduino 0107
File Edit Sketch Tools Help
sketch_mar14a
#include <ATX2.h>

void setup()
{
    OK(); // Wait for OK button
}

void loop()
{}
```

ATX2, ATMega644P @ 20 MHz on COM1

รูปแบบของโปรแกรม Arduino

The screenshot shows the Arduino IDE interface. The title bar says "sketch_mar14". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for upload, download, and other functions. The main code area contains:

```
#include <ATX2.h>

void setup()
{
    OK(); // Wait for OK button
}

void loop()
{
```

void setup()

{

สำหรับกำหนดค่า เกิดขึ้นครั้งเดียว

}

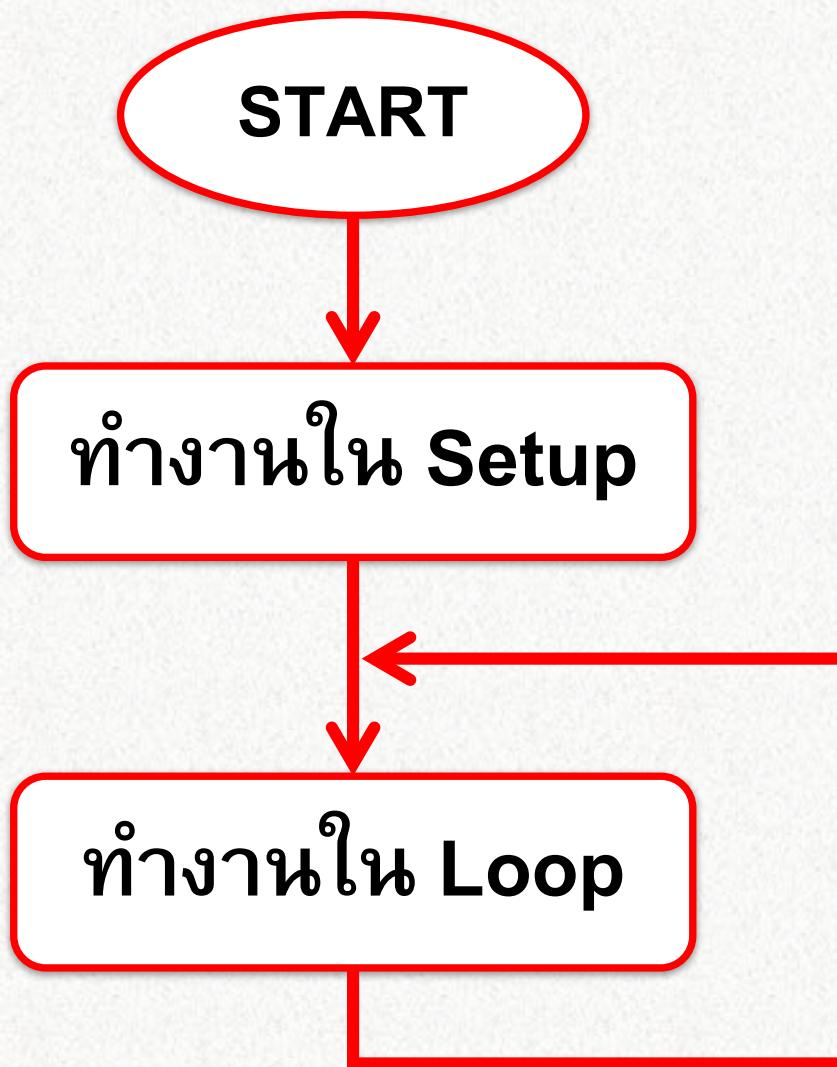
void loop()

{

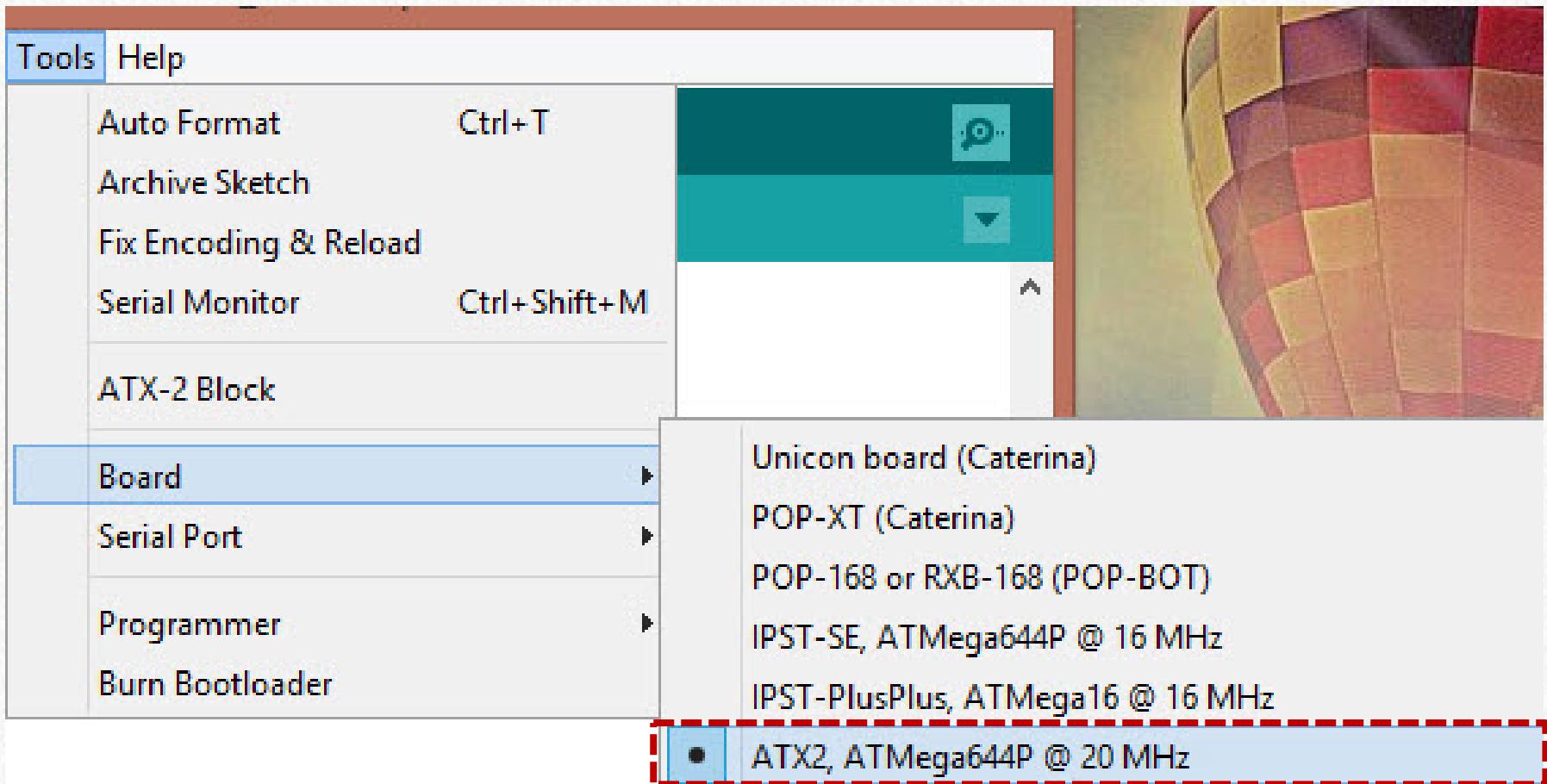
โปรแกรมหลักทำงานต่อเนื่อง

}

รูปแบบของโปรแกรม Arduino



ตรวจสอบว่าเลือกบอร์ด ATX2 แล้ว



แบตเตอรี่และขั้วต่อ



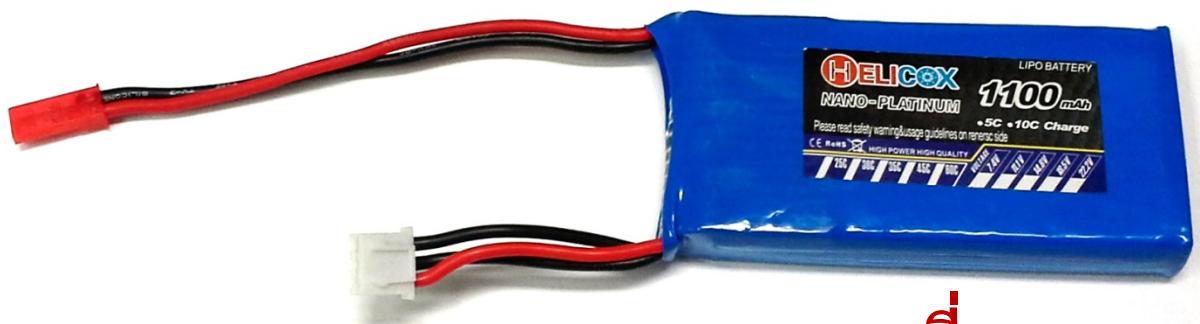
แดง ขาว
ดำ ลง

สายเชื่อมต่อแบตเตอรี่

2 เชล 7.4V

กระแส 1100mA

จ่ายกระแส 30 เท่า
ชาร์จ 5 เท่า



แบตเตอรี่ Li-Po

รายละเอียดแบตเตอรี่ ลิเธียม-โพลิเมอร์

2 เชล 7.4V

กระแส 1100mAh
จ่ายกระแส 30 เท่า
ชาร์จ 5 เท่า



1 เชล 3.7V อนุกรมกัน 2 เชล = 7.4V

จ่ายไฟ 1100 mA ต่อเนื่องได้ประมาณ 1 ชั่วโมง

จ่ายกระแสชั่วขณะได้ $1100 \times 30 = 33000mA$ O_o!

ชาร์จได้ 5 เท่า $1100 \times 5 = 5500mA$ ใช้เวลาประมาณ 20 นาที

ข้อดีของแบตเตอรี่ Li-Po



ข้อดีของแบตเตอรี่แบบ Li-Po เมื่อนำมาใช้กับหุ่นยนต์

1. มีน้ำหนักเบาในเมื่อเทียบกับความจุ (mAh)
2. จ่ายกระแสได้มากกว่าความจุ ทำให้หุ่นยนต์มีความเร็วเพิ่มขึ้นชัดเจน
3. แรงดันคงที่ หุ่นยนต์ทำงานนิ่งตลอด จนหมดความจุ
4. มีหลายรูปแบบขนาด ทำให้ยืดติดตั้งได้ง่าย
5. คายประจุด้วยตัวเอง(Self Discharge) น้อย
6. ชาร์จเต็มเร็วมาก

ข้อเสียของแบตเตอรี่ Li-Po



ข้อเสียของแบตเตอรี่แบบ Li-Po เมื่อนำมาใช้กับหุ่นยนต์

1. มีราคาแพงเมื่อเทียบกับแบตเตอรี่ชนิดอื่น ๆ
2. ต้องใช้เครื่องชาร์จที่มีความเฉพาะ ซึ่งบางแบบก็มีราคาแพง
3. ต้องดูแลเป็นพิเศษ ถ้าเกิดการลัดวงจรจะเกิดความเสียหายใหญ่หลวง
4. ต้องดูแลเรื่องความชื้น ถ้าใกล้หมุด จะเกิดการสูญเสียแรงดันและแบตเตอรี่เกิดความเสียหาย ต้องมีการตรวจวัดความชื้นของแบตเตอรี่อยู่เสมอเมื่อใช้งาน
5. เมื่อไม่ใช้งานนาน ๆ ต้องไม่ให้แบต มีความชื้นสูดเต็มค้างไว้ ไม่งั้นแบตจะบวม

วัด Volt เพื่อป้องกันแบตเสื่อม

แสดงไฟและเตือน

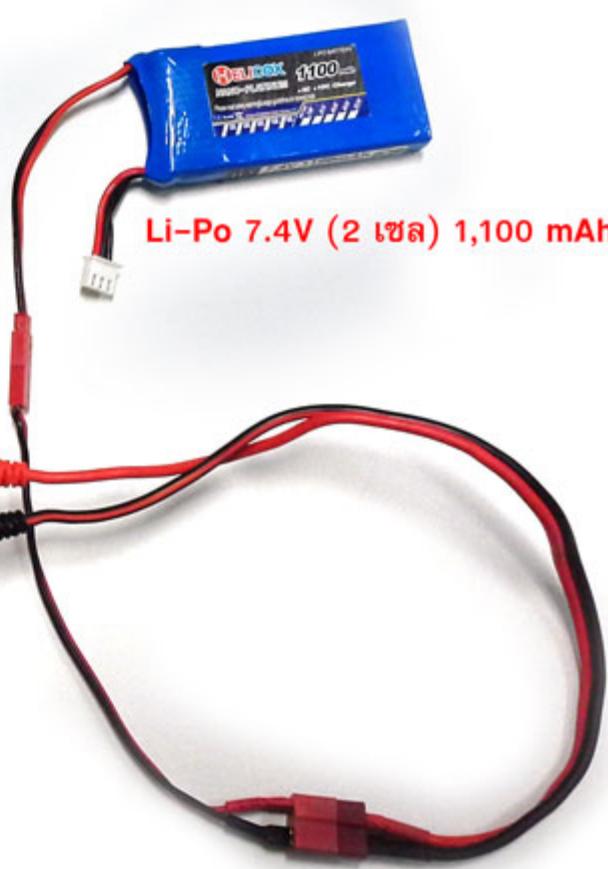


วัด Volt อย่างเดียว

วัดโวลต์และเตือน

เครื่องชาร์ตแบตกระถ่ำสูง

220V

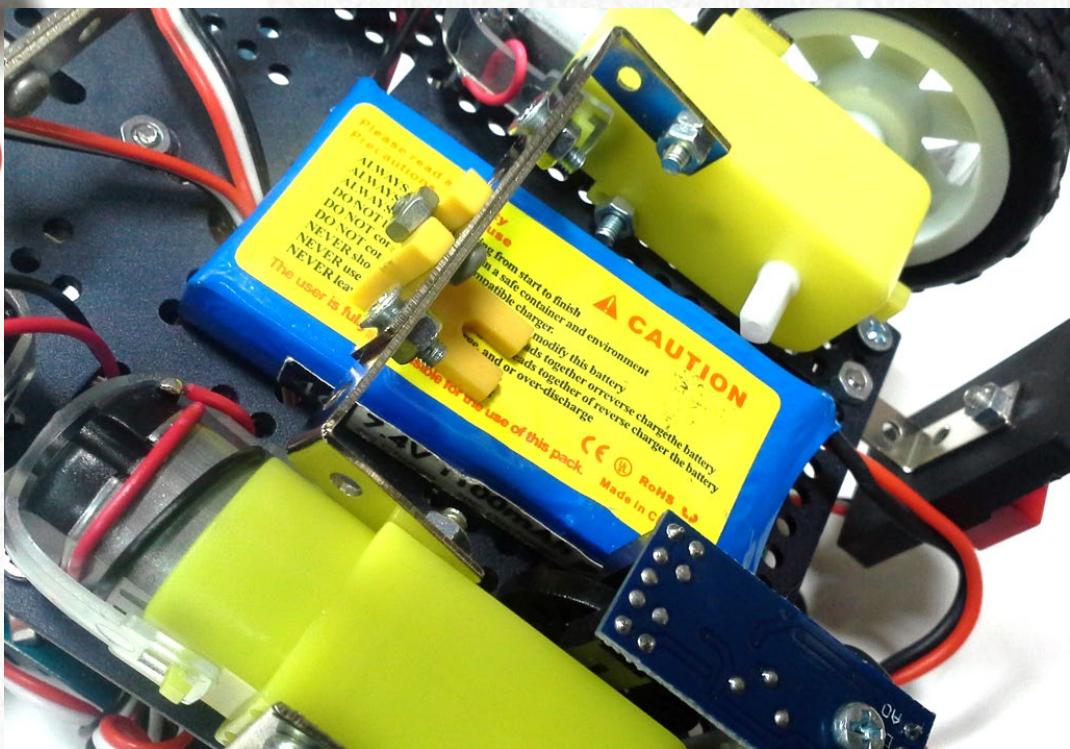
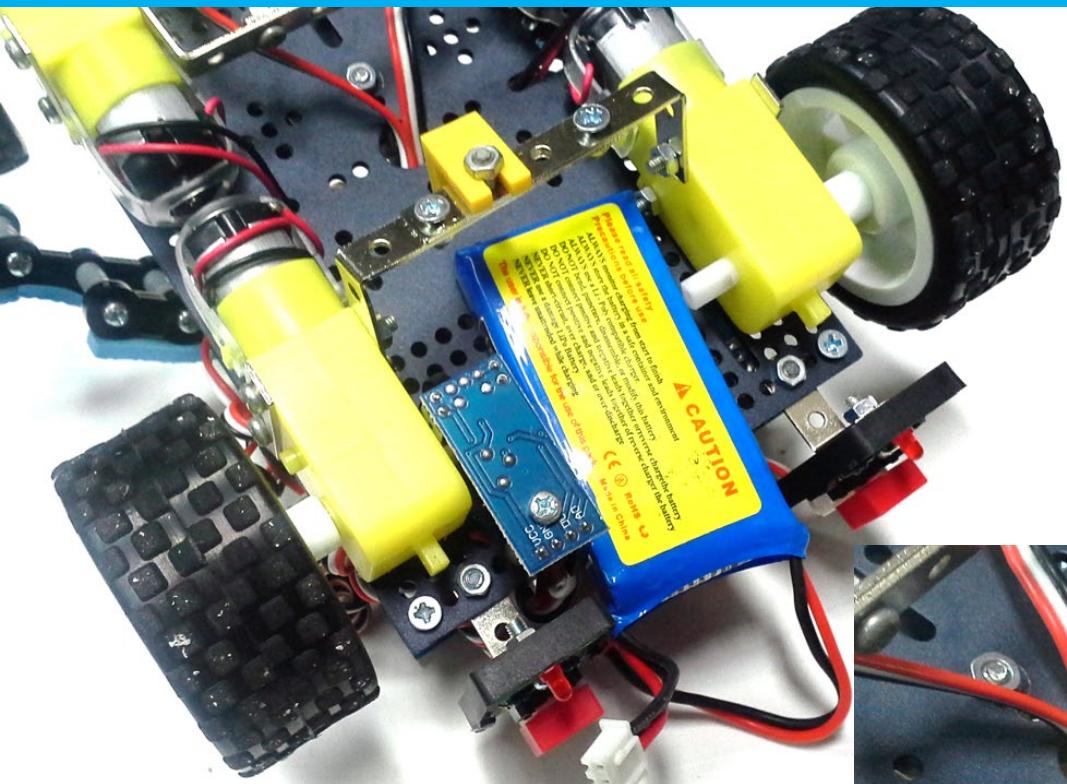


ชาร์จถ่าน Ni-MH

ชาร์จแบต Li-Po

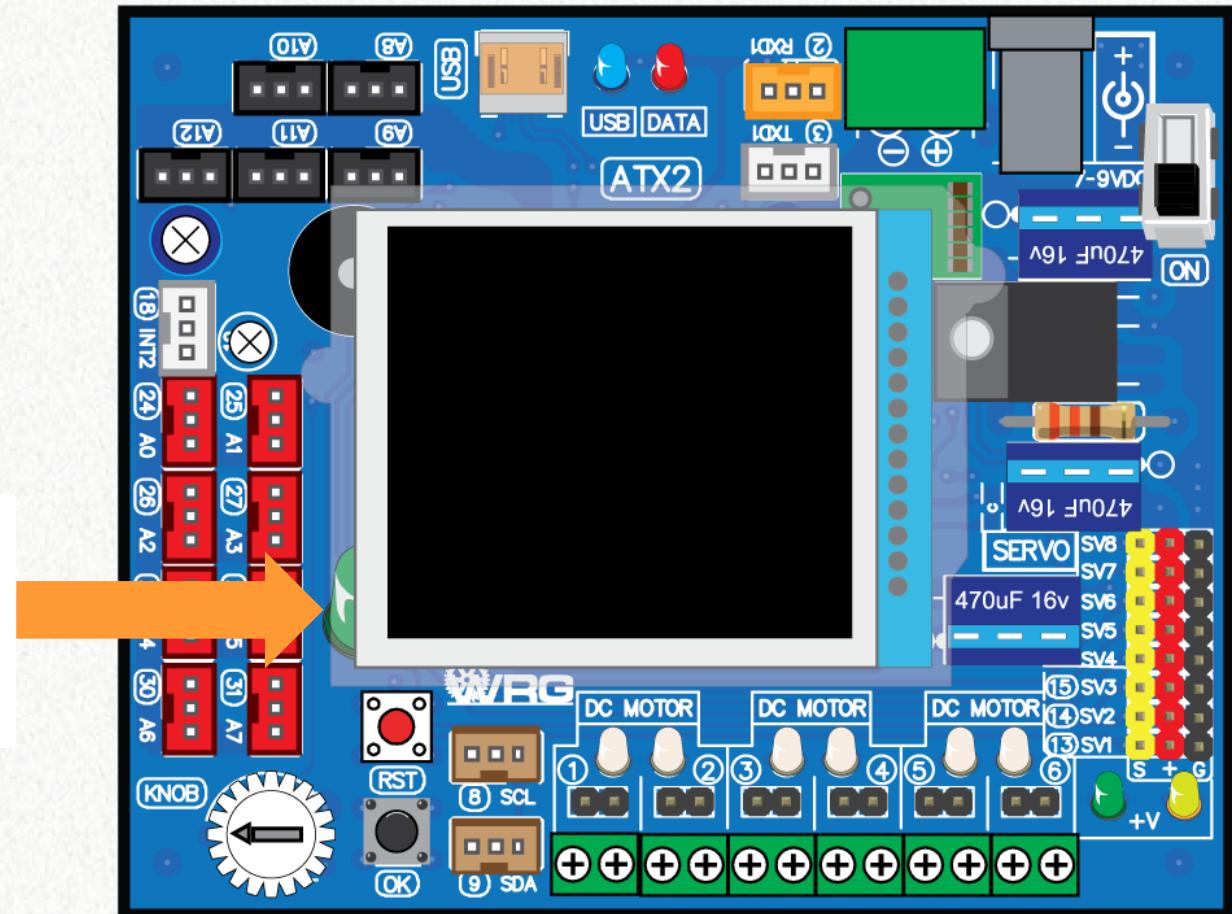
ชาร์จแบตรถยนต์

ຕິດຕັ້ງແບຕເຕອຣີກັບຫຸ່ນຍົນຕ

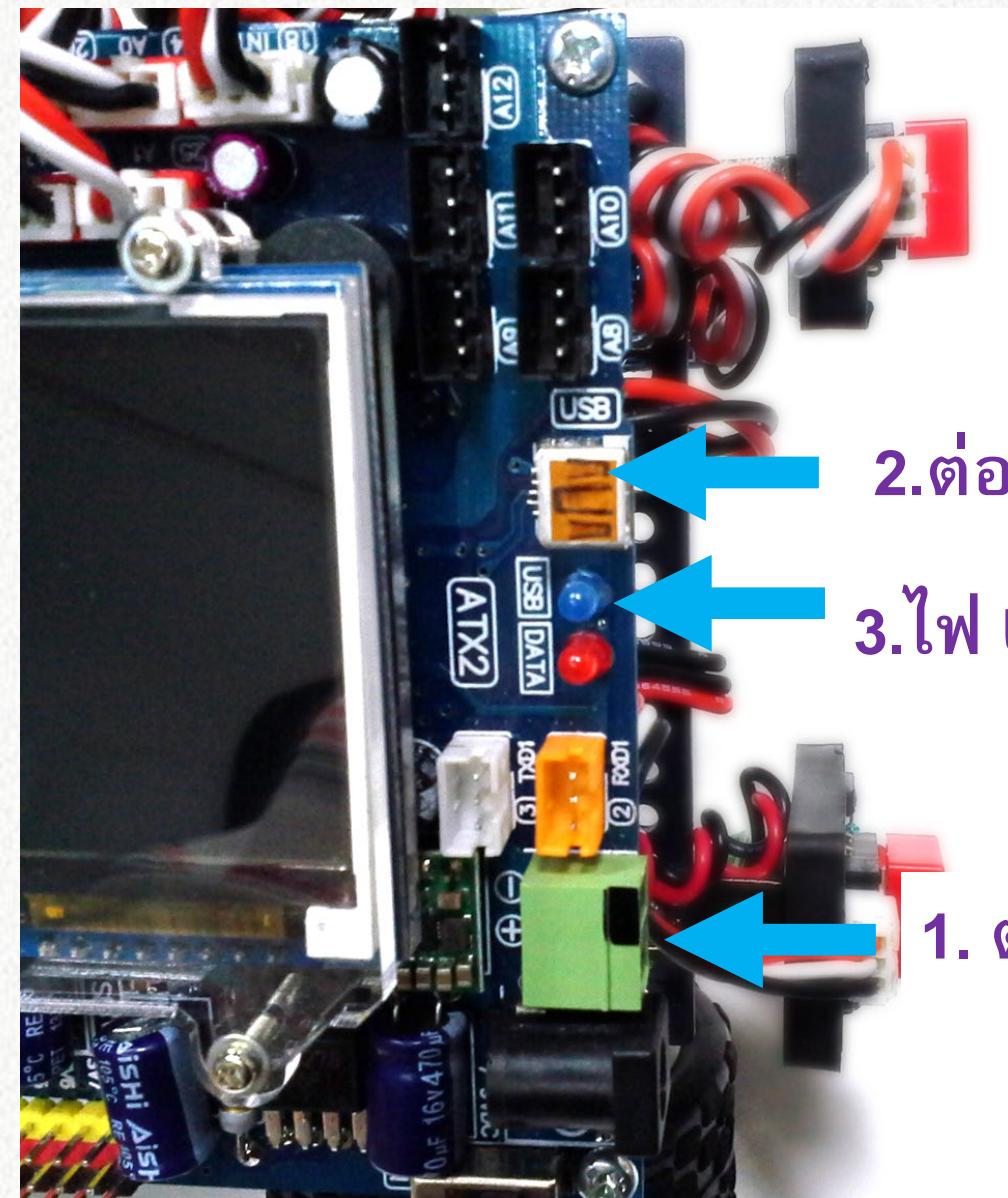


สถานะแบตเตอรี่

ถ้า LED ติด ต้อง
เปลี่ยนแบตทันที



ขั้นตอนการเชื่อมต่อ

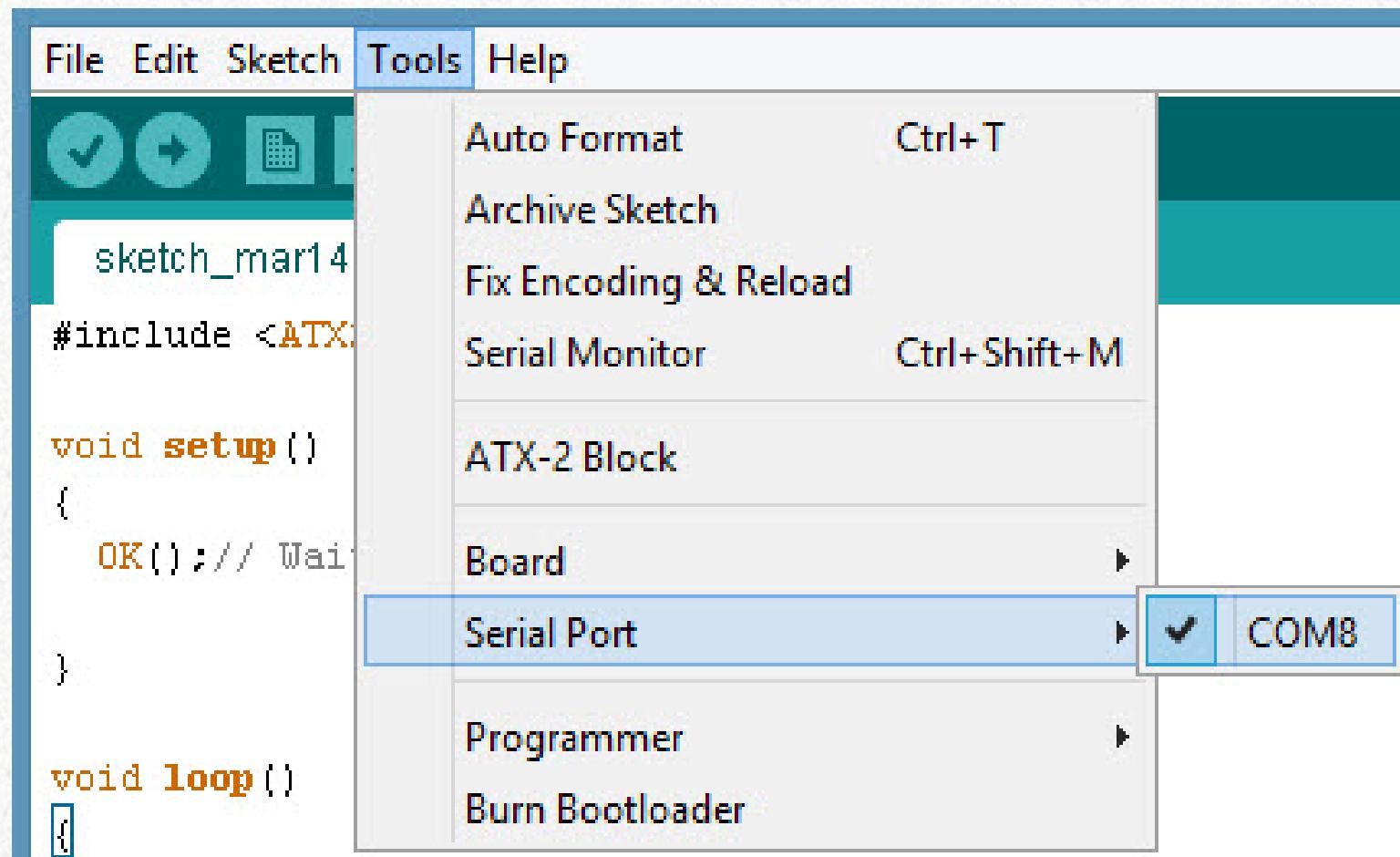


2. ต่อสาย USB กับคอมพิวเตอร์

3. ไฟ USB ติดสว่าง

1. ต่อแบตเตอรี่

เลือกพอร์ตอนุกรม



ทดสอบโปรแกรมแรก

```
#include <ATX2.h>
void setup() {
    glcd(0,0,"Hello World");
}
void loop() {
}
```

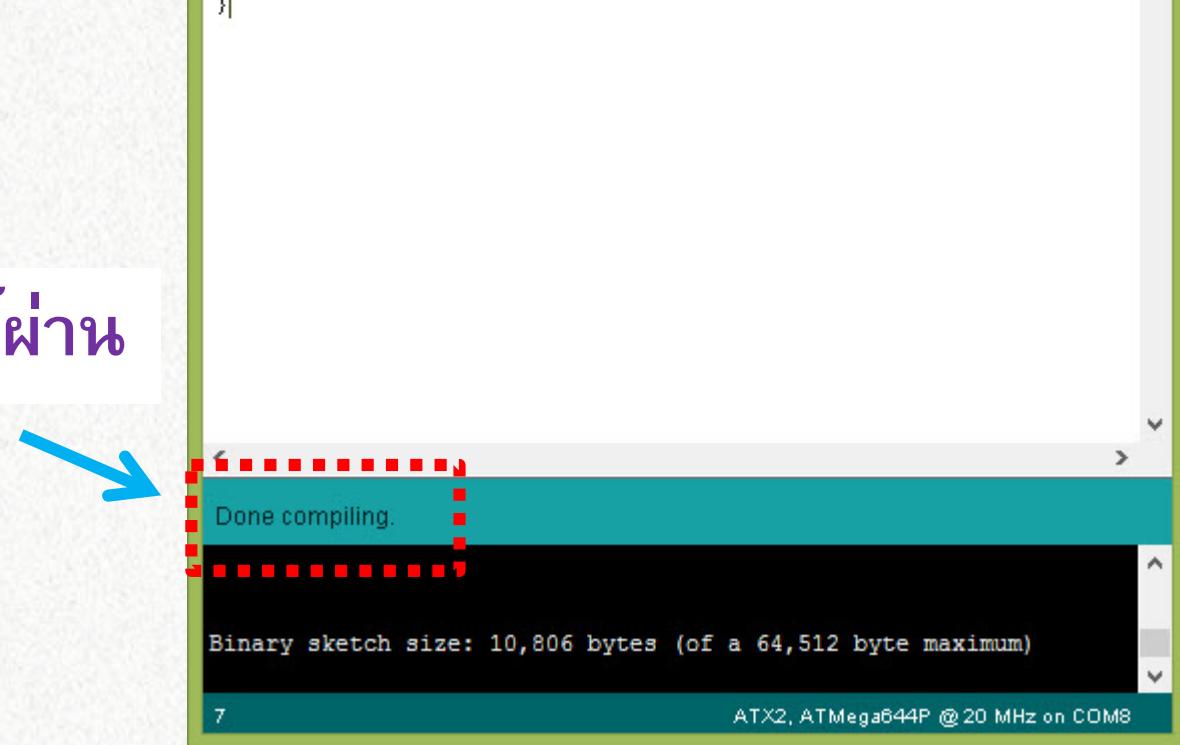
ตรวจสอบไวยกรณ์/อัพโหลด

3. อัพโหลดไปยัง ATX2

1. คอมpile



```
#include <ATX2.h>
void setup(){
    glcd(0,0,"Hello World");
}
void loop(){}
```



Done compiling.

Binary sketch size: 10,806 bytes (of a 64,512 byte maximum)

ATX2, ATmega644P @ 20 MHz on COM8

2. แจ้งว่าคอมpileผ่าน

ผลลัพธ์ที่บอร์ด ATX2



คุณสมบัติของจอ GLCD

21 ตัวอักษร

16 บรรทัด

กว้าง 128
สูง 160



คำสั่งของ GLCD

glcd แสดงข้อความที่จอ GLCD ได้ 21 ตัว 16 บรรทัด (size 1)

รูปแบบ

void glcd(x, y, *p, ...)

พารามิเตอร์

- x** คือตำแหน่งบรรทัดมีค่าตั้งแต่ 0-15
- y** คือตำแหน่งตัวอักษร มีค่าตั้งแต่ 0-24
- *p** คือข้อความที่ต้องการนำมาแสดง

ค่าพิเศษ

%d แสดงตัวเลขจำนวนเต็มในช่วง -32,768 ถึง 32,767

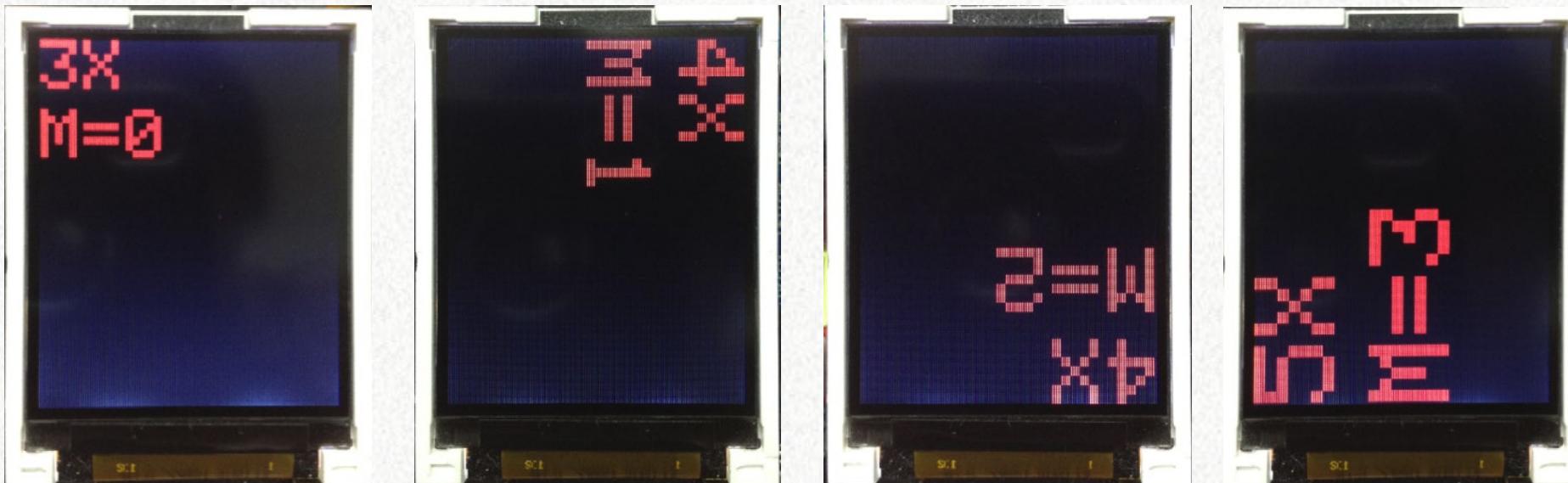
%h แสดงตัวเลขฐานสิบหก

%b แสดงตัวเลขฐานสอง

%l แสดงตัวเลขจำนวนเต็มในช่วง -2,147,483,648 ถึง 2,147,483,647

%f แสดงผลตัวเลขจำนวนจริง (แสดงทศนิยม 3 หลัก)

คำสั่งของ glcdMode (หมุนหน้าจอ)



ปกติเป็น Mode 0

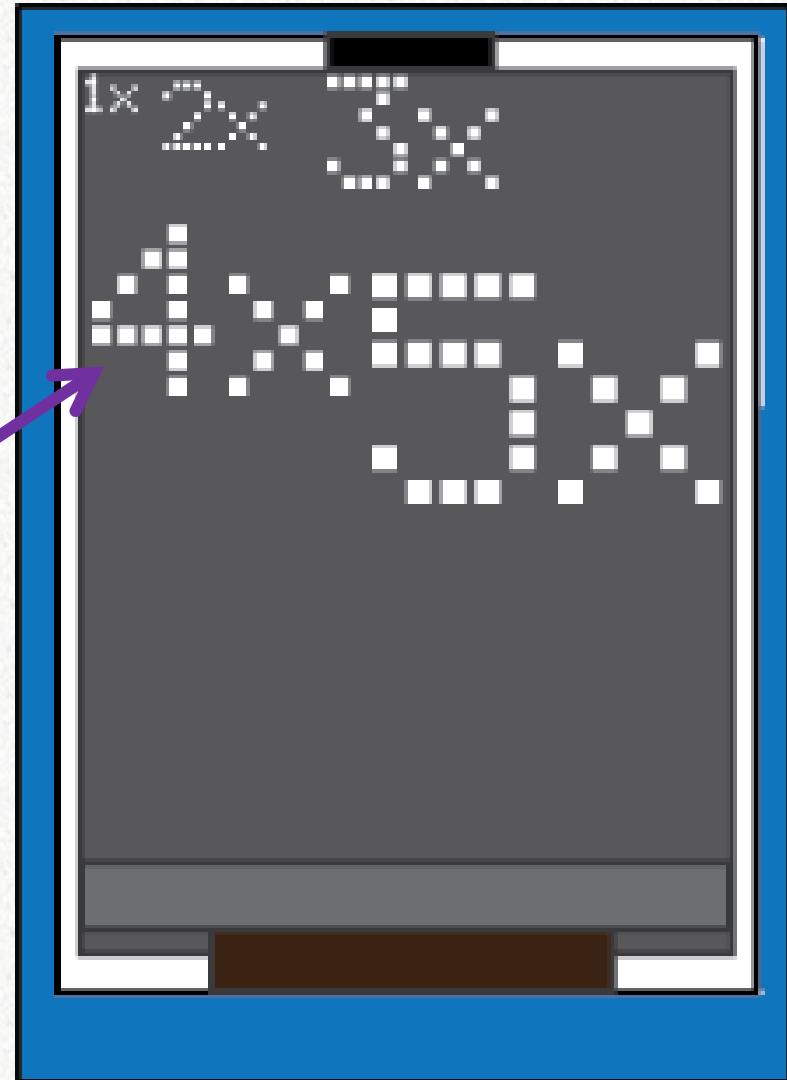
ตัวอย่าง : **glcdMode(1);**

ปรับขนาดตัวอักษร

setTextSize()

ขนาดตัวอักษรเป็น 4 เท่าขนาดปกติ

ตัวอย่าง : **setTextSize(4);**



ค่าสีตัวอักษร

ตัวอย่าง

```
#include <ATX2.h>
void setup() {
    setTextColor(GLCD_WHITE);
    lcd(0,0,"Hello");
    setTextColor(GLCD_GREEN);
    lcd(1,0,"World");
}
void loop() {}
```



setTextColor (COLOR)

GLCD_RED,
GLCD_GREEN,
GLCD_BLUE,
GLCD_YELLOW,
GLCD_BLACK,
GLCD_WHITE,
GLCD_CYAN,
GLCD_MAGENTA
GLCD_ORANGE
GLCD_LIME
GLCD_VIOLET
GLCD_PINK
GLCD_DOLLAR
GLCD_SKY
GLCD_BROWN
GLCD_DARKGREEN
GLCD_NAVY
GLCD_GRAY
GLCD_DARKGRAY

ค่าสีพื้นหลังตัวอักษร

ตัวอย่าง

```
#include <ATX2.h>
void setup() {
    setTextBackgroundColor(GLCD_RED);
    setTextColor(GLCD_YELLOW);
    glcd(0,0,"Hello World");
}
void loop() {}
```



setTextBackgroundColor (COLOR)

แสดงรูปทรงเรขาคณิต

glcdRect (x, y, width, height, color)
glcdFillRect (x, y, width, height, color)
glcdCircle (x, y, radius, color)
glcdFillCircle (x, y, radius, color)
glcdLine (x1, y1, x2, y2, color)

x ตำแหน่งแนวนอน

y ตำแหน่งแนวตั้ง

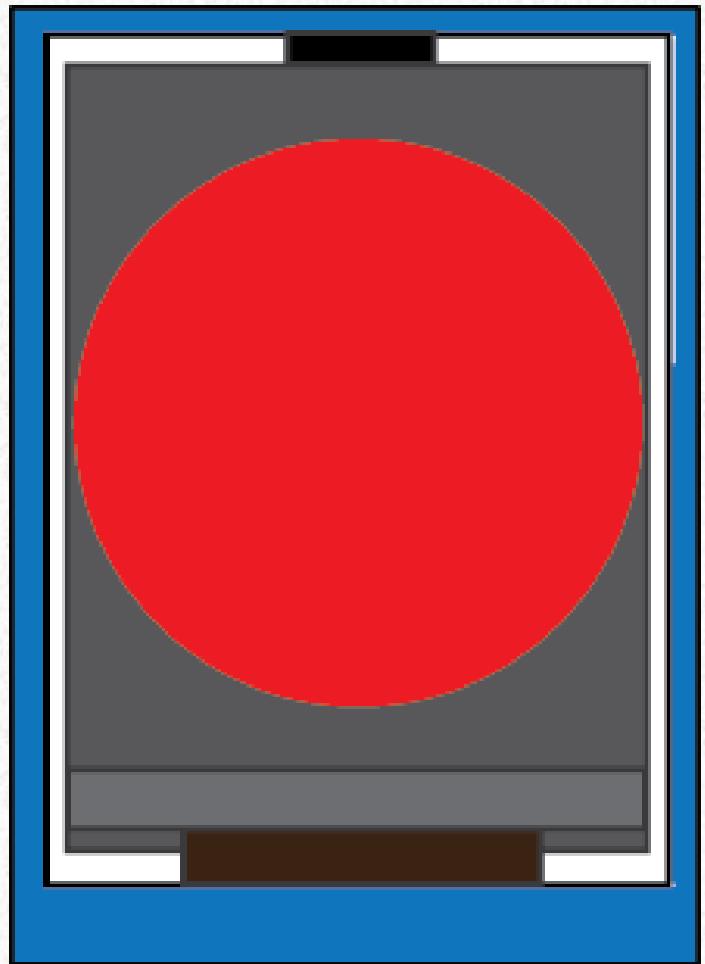
width ความกว้าง

height ความสูง

radius รัศมี

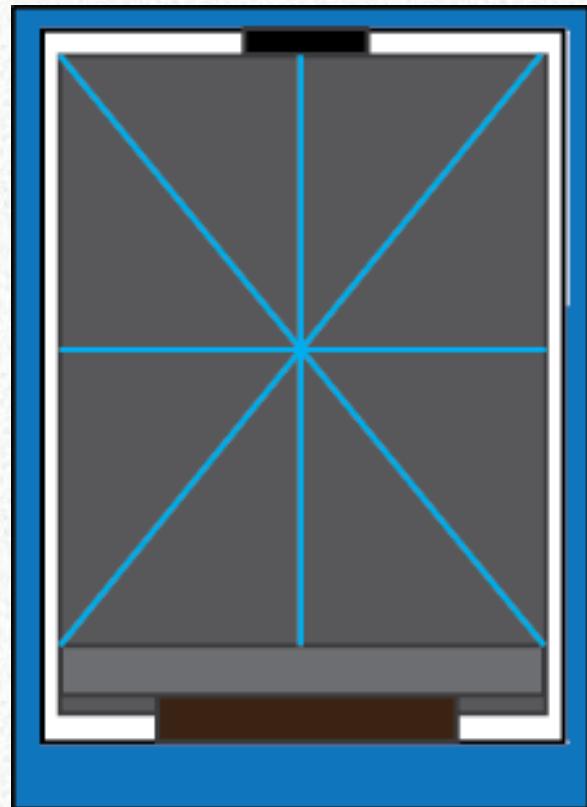
color สี

บททดสอบ 1



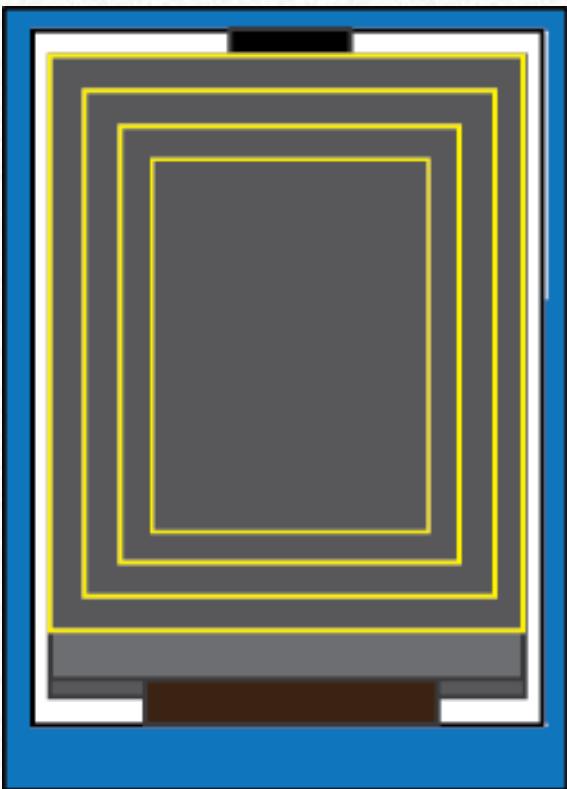
สร้างวงกลมสีแดงอยู่กึ่งกลางจอภาพ รัศมีเต็มจอพอดี

ลากเส้น 4 เส้นโดยมีจุดตัดอยู่กลางจณาพอดี



บทที่ 3

สร้างสีเหลี่ยมช้อนกันด้งรูป



คำสั่งของ GLCD

glcd แสดงข้อความที่จอ GLCD ได้ 21 ตัว 16 บรรทัด (size 1)

รูปแบบ

void glcd(x, y, *p, ...)

พารามิเตอร์

- x** คือตำแหน่งบรรทัดมีค่าตั้งแต่ 0-15
- y** คือตำแหน่งตัวอักษร มีค่าตั้งแต่ 0-24
- *p** คือข้อความที่ต้องการนำมาแสดง

ค่าพิเศษ

%d แสดงตัวเลขจำนวนเต็มในช่วง -32,768 ถึง 32,767

%h แสดงตัวเลขฐานสิบหก

%b แสดงตัวเลขฐานสอง

%l แสดงตัวเลขจำนวนเต็มในช่วง -2,147,483,648 ถึง 2,147,483,647

%f แสดงผลตัวเลขจำนวนจริง (แสดงทศนิยม 3 หลัก)

การแสดงผลค่าตัวเลข

glcd(0, 0, "%d", 100);



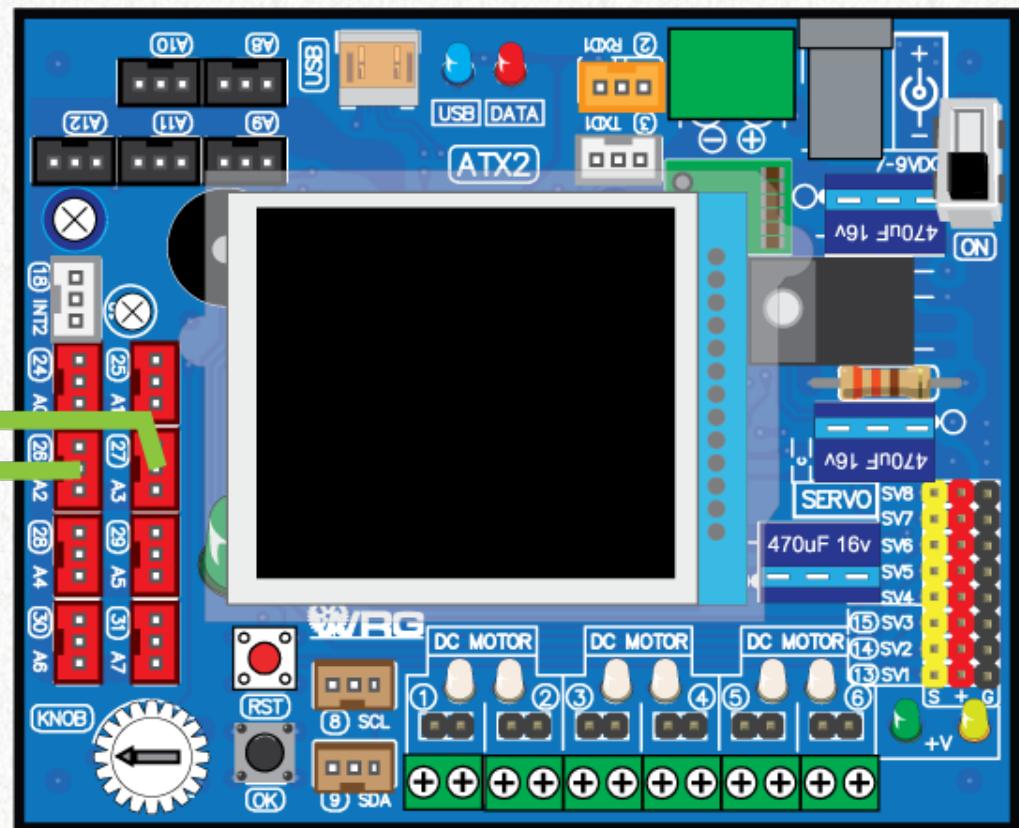
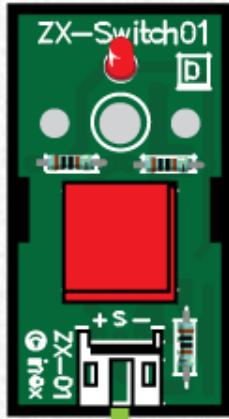
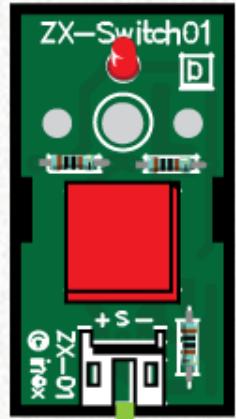
ค่าตัวแปรที่ใช้งานบ่อย ๆ

byte	0-255	(unsigned char)
word	0-65535	(unsigned int)
boolean	0-1	True False
int	-32768	ถึง 32767
char	-128	ถึง 127
float	-3.4×10^{38}	ถึง 3.4×10^{38}

หาข้อมูลเพิ่มเติมจาก **reference**

ATX2 กับ ZX-Switch01

ต่อ กับขา 26 ต่อ กับขา 27



out(ch, state);

ส่งค่าสถานะ(state) 0 หรือ 1

ออกไปยังตำแหน่งขา (ch) ที่ระบุ

เช่น out(17,1);

คำสั่งส่งค่าออกເອາດ්පුජිජෙල

ATX2 กับ ลำโพงเปียโซ

- ใช้ลำโพงเปียโซ มีอิมพีเดนซ์ 32W
- มีค่าความถี่ย่าน 300Hz ถึง 3000 Hz

beep () ;

กำหนดเสียงความถี่ 500 Hz นาน 0.1 วินาที

sound(freq, time) ;

freq ความถี่เสียง

time เวลาที่เสียงดัง



ตัวอย่าง : สร้างเสียงด้วย 1 วินาที

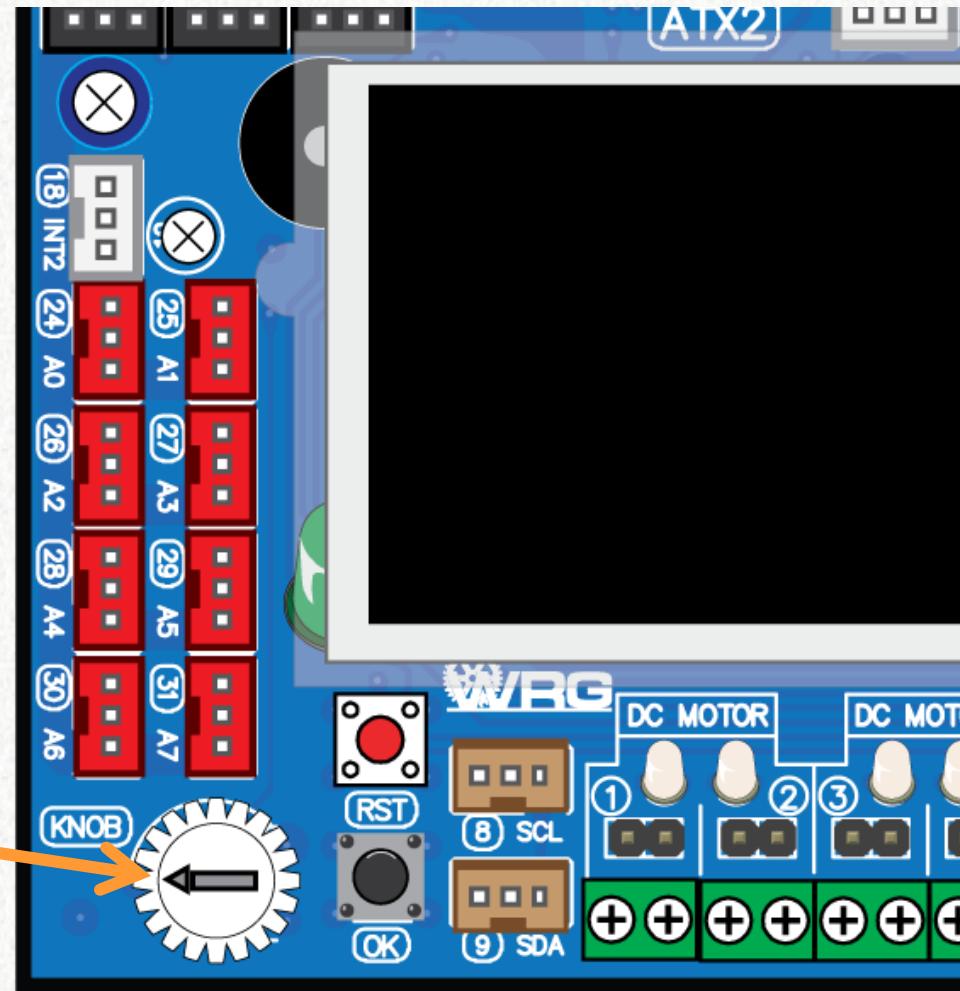
```
#include <ATX2.h>
void setup() {
}
void loop() {
    beep();
    delay(1000);
}
```

ตัวอย่าง : สร้างเสียงความถี่ 1200 Hz นาน 0.5 วินาที

```
#include <ATX2.h>
void setup() { }
void loop() {
    sound(1200, 500);
    sleep(1000);
}
```

knob() : ตัวต้านทานปรับค่าได้

หมุนเพื่อปรับค่า 0-1000



การใช้งานคำสั่ง knob()

knob() ใช้อ่านค่าตัวต้านทานปรับค่าได้บน ATX2

ค่าอยู่ในช่วง 0-1000

รูปแบบ

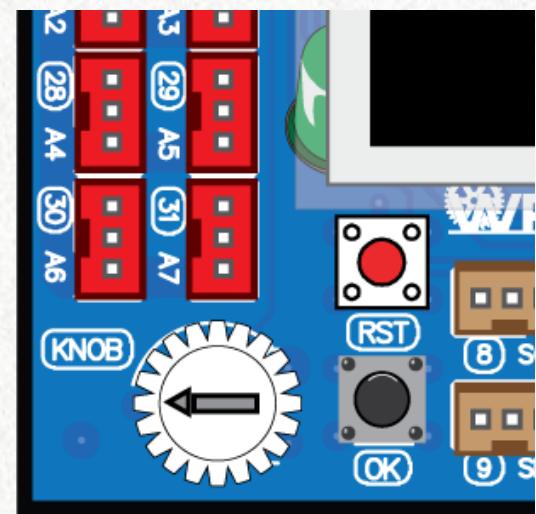
knob () ;

การคืนค่า

ค่าที่อ่านได้จาก knob มีค่าระหว่าง 0-1000

ตัวอย่าง

glcd(1,1,"%d",knob()) ;



การใช้งานคำสั่ง knob(x)

รูปแบบ

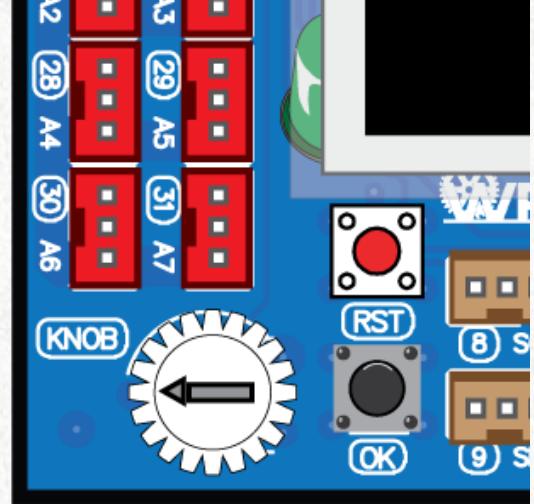
knob (x) ;

x คือค่า Scale หรือช่วงที่ต้องการ

ตัวอย่าง

glcd(1,1,"%d",knob(180));

หน้าจอจะแสดงค่า 0-180 เพราะตั้งค่าสเกลเป็น 180



การใช้งานคำสั่ง knob(x,y)

รูปแบบ

knob (x ,y) ;

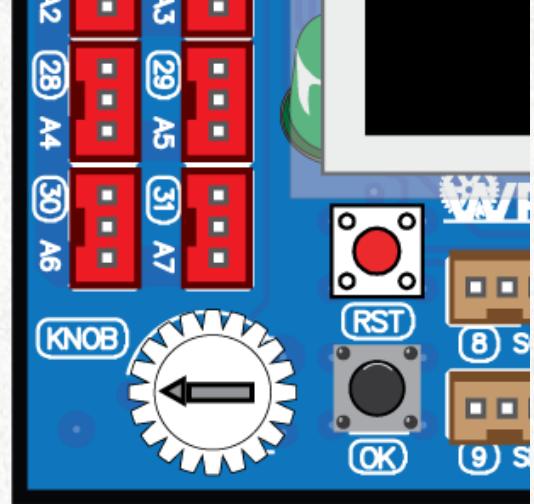
x คือค่า Scale ช่วงเริ่มต้น

y คือค่า Scale ช่วงท้าย

ตัวอย่าง

glcd(1,1,"%d",knob(10,90));

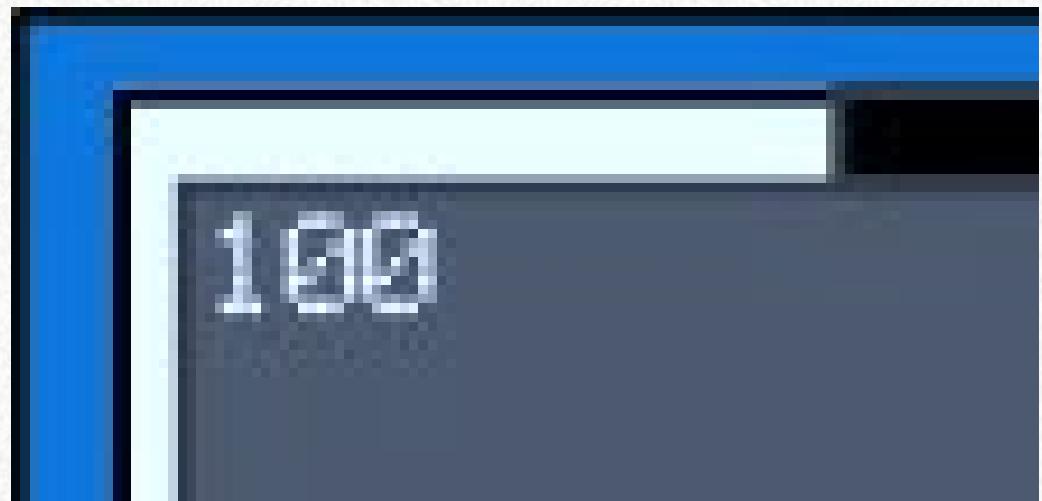
หน้าจอจะแสดงค่า 10-90 ตามการหมุน knob()



การแสดงผลค่า knob() ที่จอ GLCD



```
glcd(0, 0, "%d", knob());
```

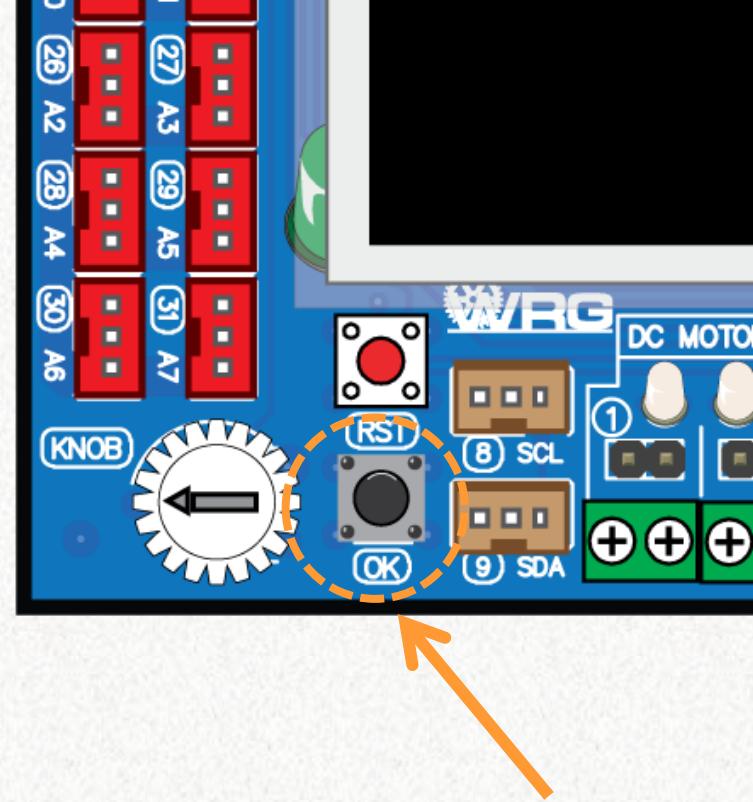


คำสั่ง sw_OK()

กดเป็นจริง
ไม่กดเป็นเท็จ

ตัวอย่าง

```
#include <ATX2.h>
void setup() {}
void loop() {
    if (sw_OK()) {
        sound(1200, 100);
    }
}
```



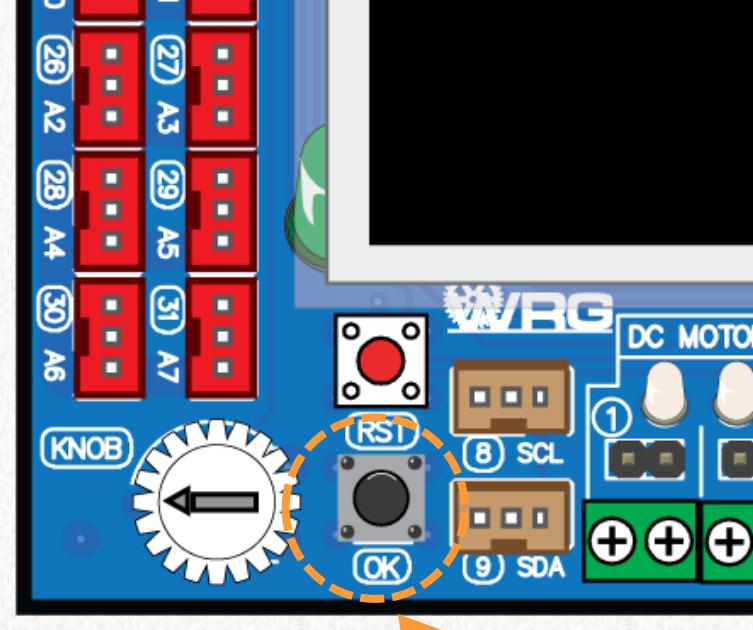
สวิตซ์ OK

คำสั่ง sw_OK_press()

รอจนกระทั่งกดสวิตช์ OK
แล้วปล่อยสวิตช์ จึงจะทำงาน

ตัวอย่าง

```
#include <ATX2.h>
void setup () { }
void loop () {
    sw_OK_press () ;
    beep () ;
}
```



สวิตช์ OK

คำสั่ง OK ()

1. แสดงข้อความที่หน้าจอ
2. รอนการะทั้งกดสวิตช์ OK
3. ทำงานคำสั่งถัดไป

ตัวอย่าง

```
#include <ATX2.h>
void setup () {
    OK () ;
}
void loop () {
}
```



คำสั่งขับมอเตอร์

motor (CH , POW)

CH 1-6 คือมอเตอร์ 1 ถึง 6

12 สำหรับมอเตอร์ 1 และ 2

34 สำหรับมอเตอร์ 3 และ 4

56 สำหรับมอเตอร์ 5 และ 6

100,ALL,ALL4 สำหรับมอเตอร์ 1 ถึง 4

106 หรือ ALL6 สำหรับมอเตอร์ทั้ง 6 ตัว

POW ความเร็ว -100 ถึง 100

ค่าบวก เดินหน้า

ค่าลบ ถอยหลัง

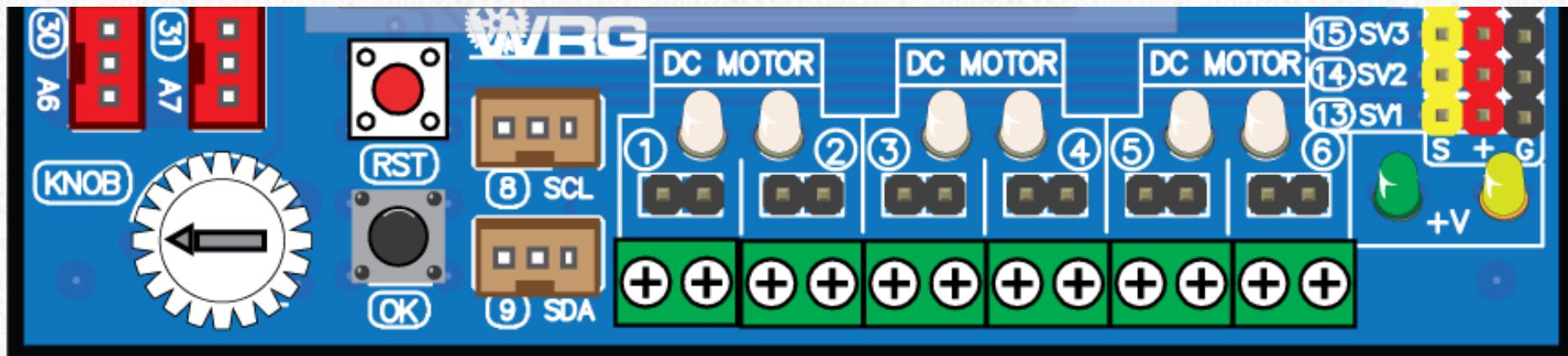
ตัวอย่าง : motor()

motor (CH , POW)

motor(1,60); ให้มอเตอร์ 1 ไปหน้าความเร็ว 60%

motor(12,-80); ให้มอเตอร์ 1 และ 2 ถอยหลังความเร็ว 80%

motor(ALL,100); ให้มอเตอร์ 1-4 ไปด้านหน้าด้วยความเร็ว 100%



คำสั่งหยุดมอเตอร์

motor_stop(CH)

CH 1-6 คือมอเตอร์ 1 ถึง 6

12 สำหรับมอเตอร์ 1 และ 2

34 สำหรับมอเตอร์ 3 และ 4

56 สำหรับมอเตอร์ 5 และ 6

100, ALL, ALL4 สำหรับมอเตอร์ 1 ถึง 4

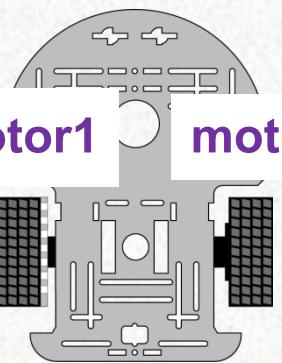
106 หรือ ALL6 สำหรับมอเตอร์ทั้ง 6 ตัว

คำสั่ง ao() เท่ากับ motor_stop(12); มอเตอร์ 1 และ 2 หยุด

คำสั่ง AO() เท่ากับ motor_stop(ALL); มอเตอร์ 1-4 หยุด

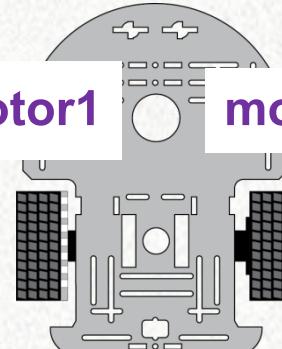
ฟังก์ชันขับเคลื่อนหุ่นยนต์ กรณีขับเคลื่อน 2 ล้อ

เดินหน้า



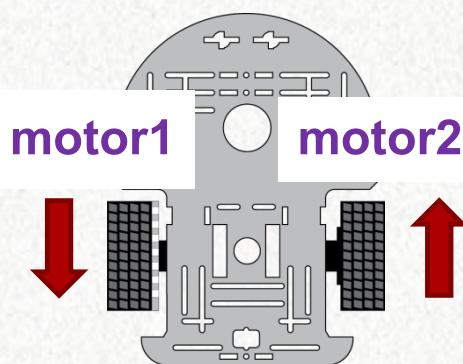
`fd (speed);`

ถอยหลัง



`bk (speed);`

เลี้ยวซ้าย



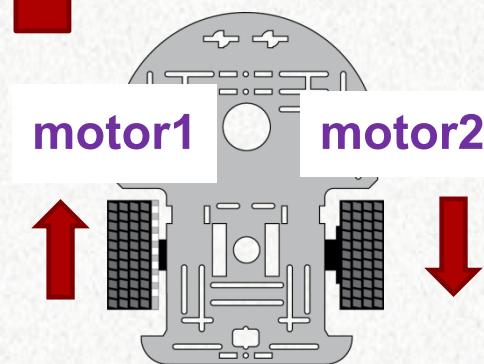
`sl (speed);`

เลี้ยวด้านเดียว

`tl (speed);`

`tr (speed);`

เลี้ยวขวา



`sr (speed);`

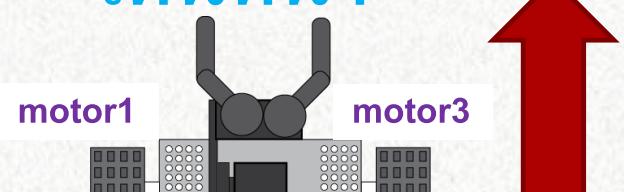
หยุดหุ่นยนต์

`ao ();`

`speed = 0-100`

ฟังก์ชันขับเคลื่อนหุ่นยนต์ กรณีขับเคลื่อน 4 ล้อ

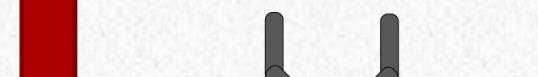
เดินหน้า



เลี้ยวซ้าย

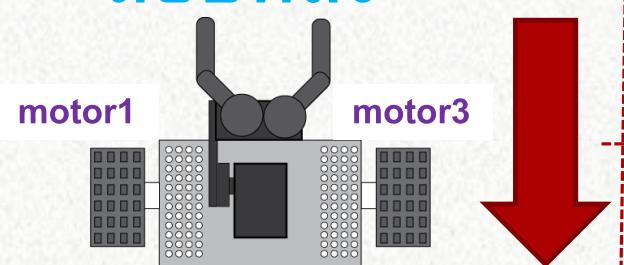


เลี้ยวขวา



FD (speed) ;

ถอยหลัง



SL (speed) ;

BK (speed) ;

หยุดหุ่นยนต์

AO () ;

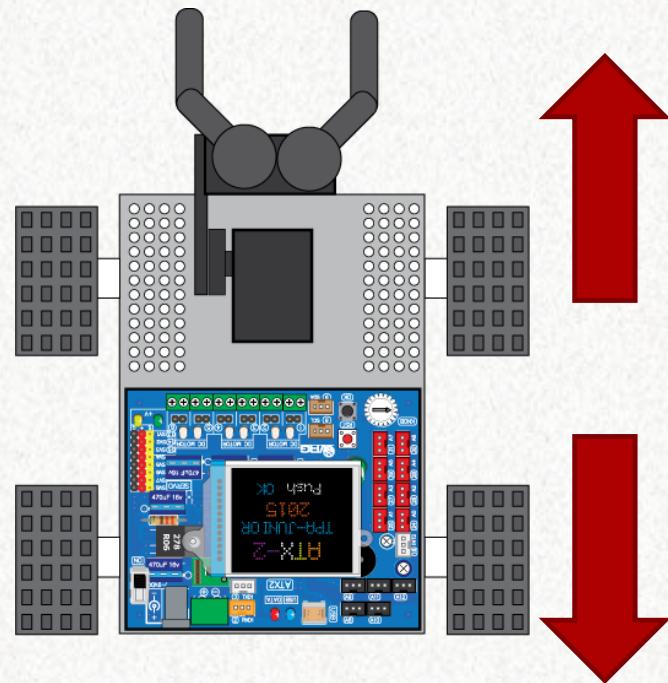
SR (speed) ;

speed = 0-100

ทดสอบขับเคลื่อน

เดินหน้า 1 วินาที ถอยหลัง 1 วินาที

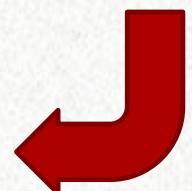
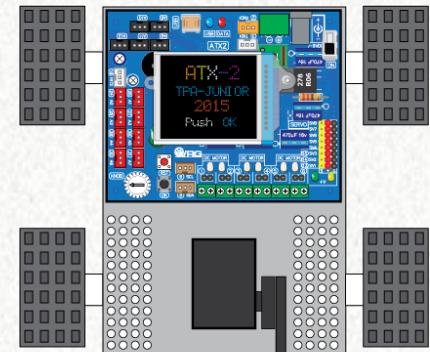
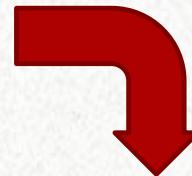
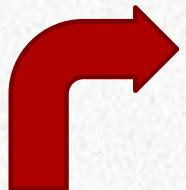
```
#include <ATX2.h>
void setup() {OK(); }
void loop() {
    FD(50); delay(1000);
    BK(50); delay(1000);
}
```



ทดสอบขับเคลื่อน

เดินวนเป็นรูปสี่เหลี่ยม

```
#include <ATX2.h>
void setup () {
    OK();
}
void loop () {
    FD(50);
    delay(1000);
    SR(50);
    delay(700);
}
```



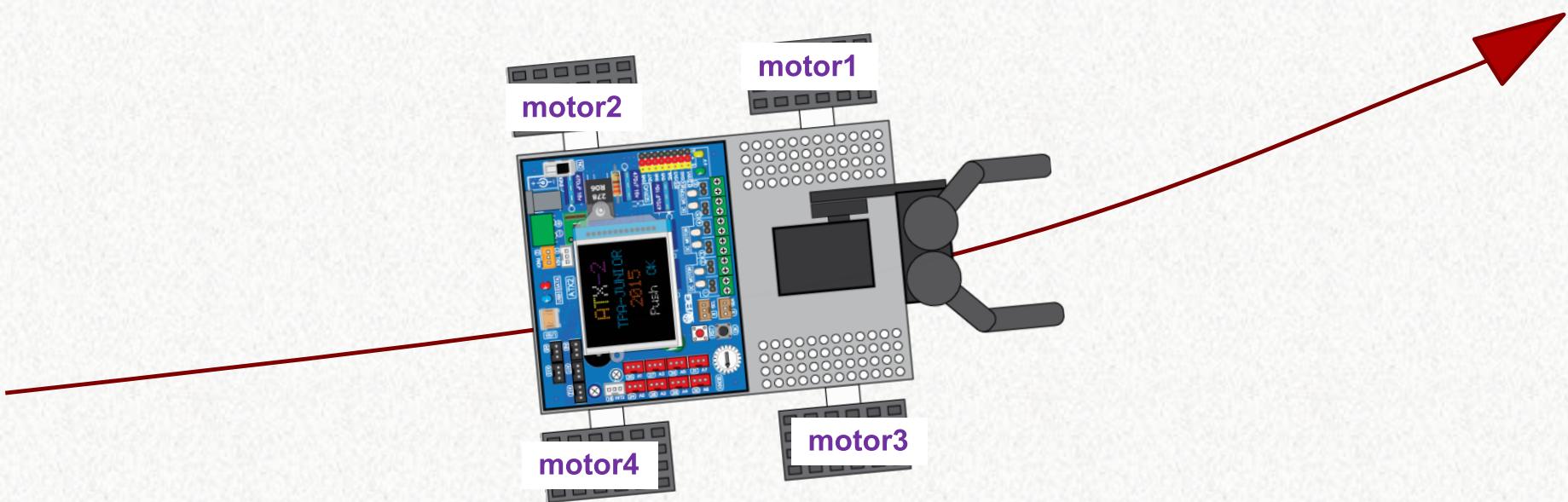
ปรับหุ่นยนต์ให้เคลื่อนที่ตรง

FD2 (Speed1, speed2) ; เดินหน้า

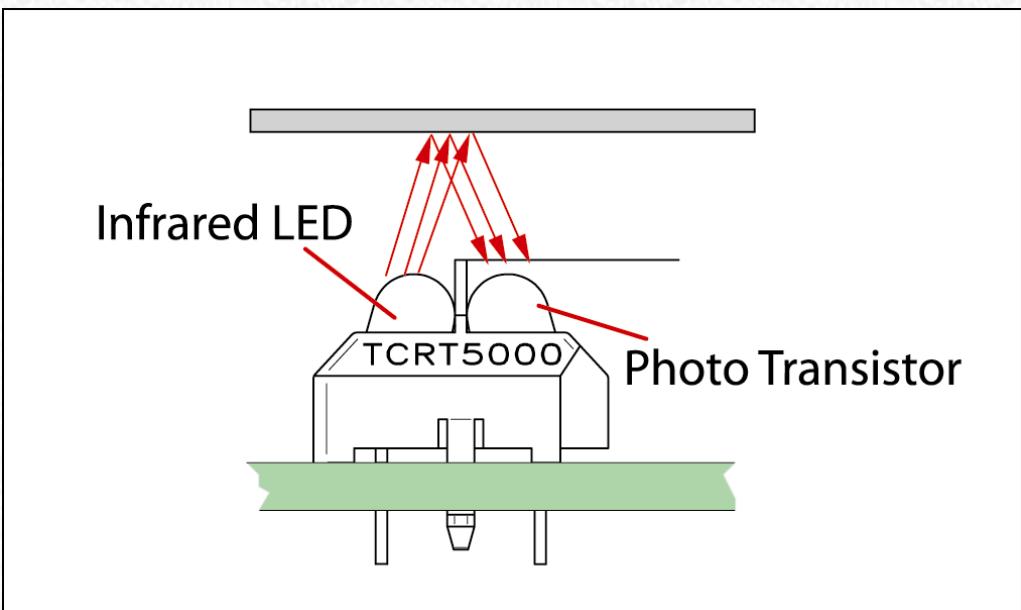
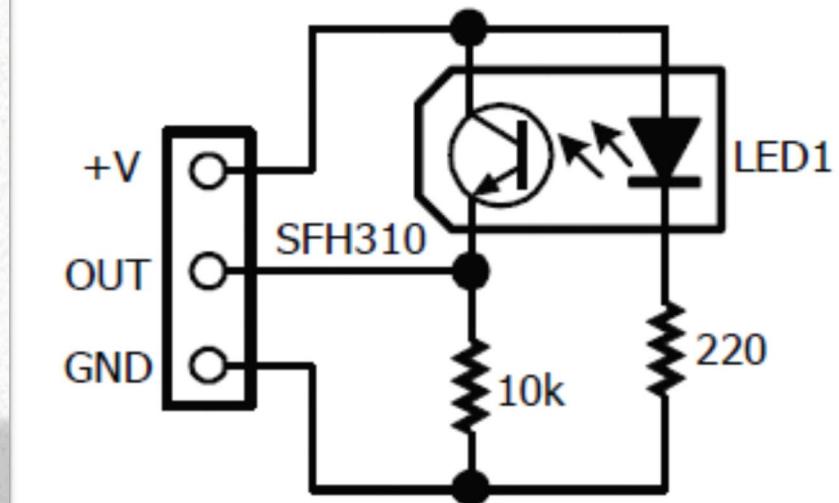
BK2 (Speed1, speed2) ; ถอยหลัง

Speed1 คือมอเตอร์ 1 และ 2

Speed2 คือมอเตอร์ 3 และ 4



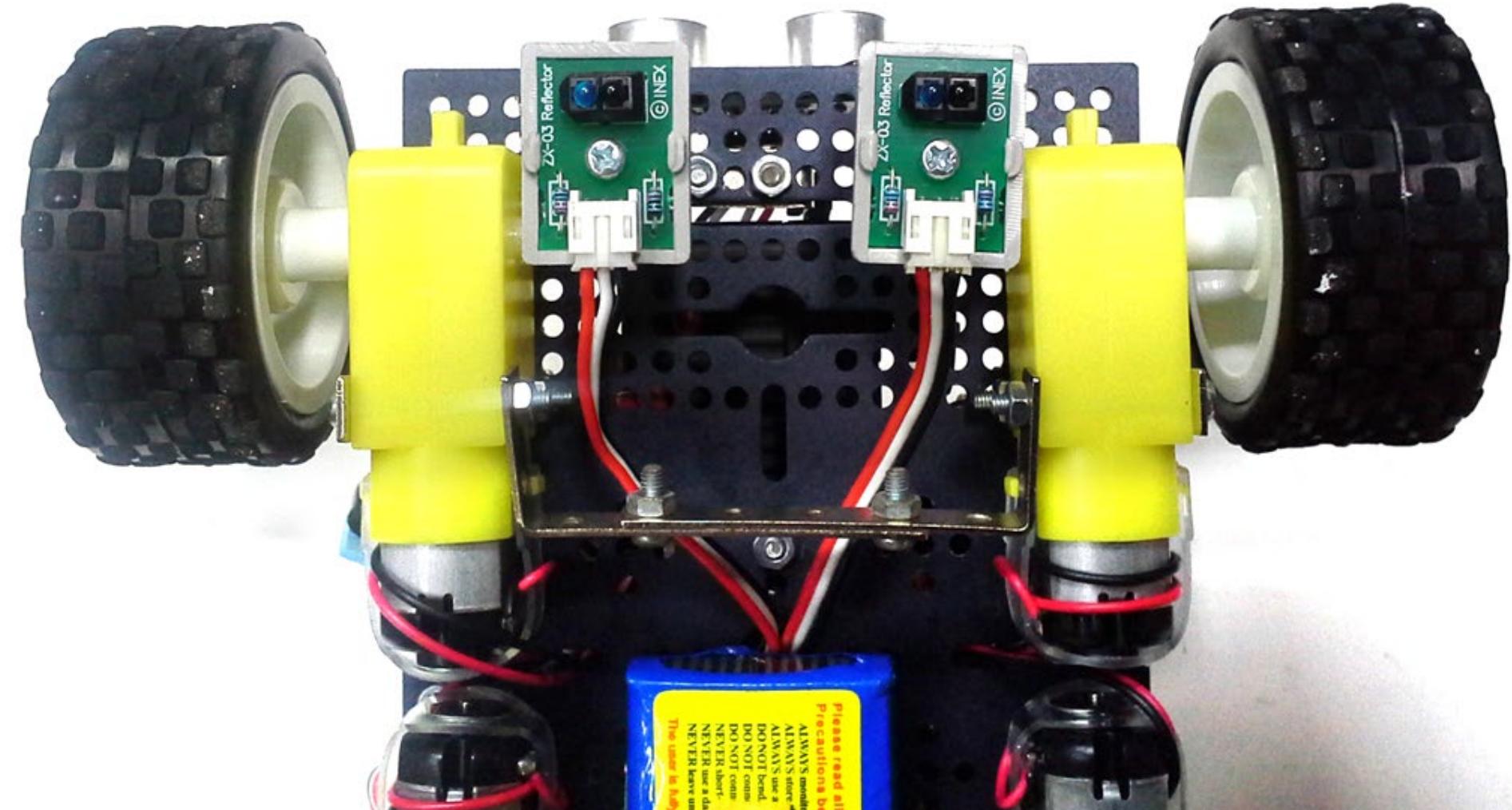
ແຜງງຈຣຕຣວຈຈັບກາຮສະຫອນ ZX-03



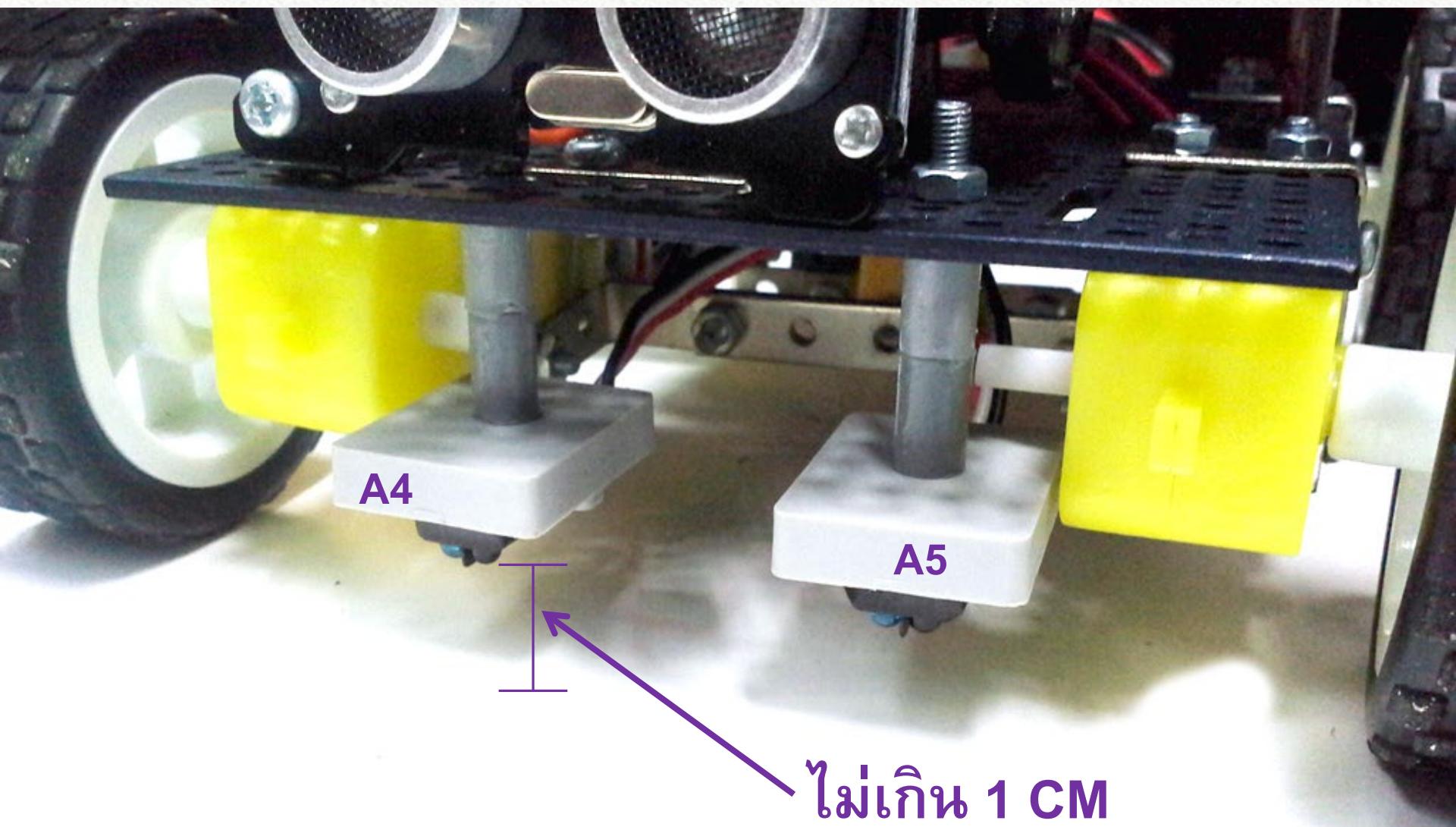
ตำแหน่งติดตั้งแผงวงจร ZX-03

A4

A5



ตำแหน่งติดตั้งแผงวงจร ZX-03

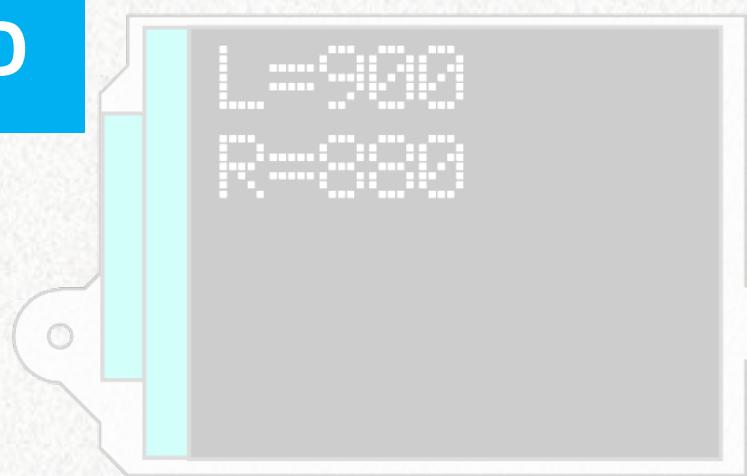


อ่านค่าจาก ZX-03 แสดงที่ GLCD

หาค่ากลาง

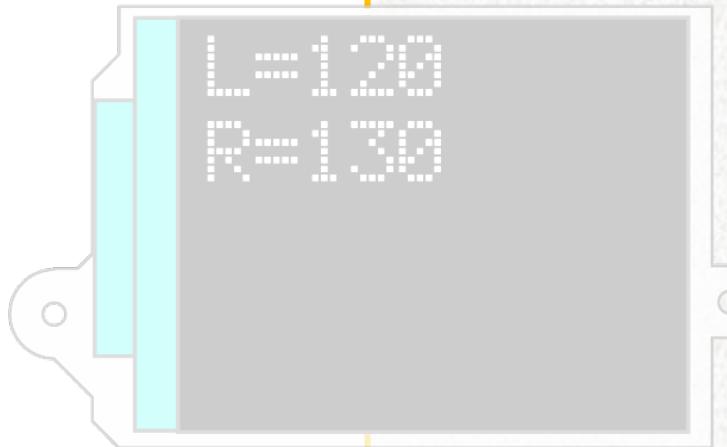
$$\text{RefL} = (900+120)/2 = 510$$

$$\text{RefR} = (880+130)/2 = 505$$



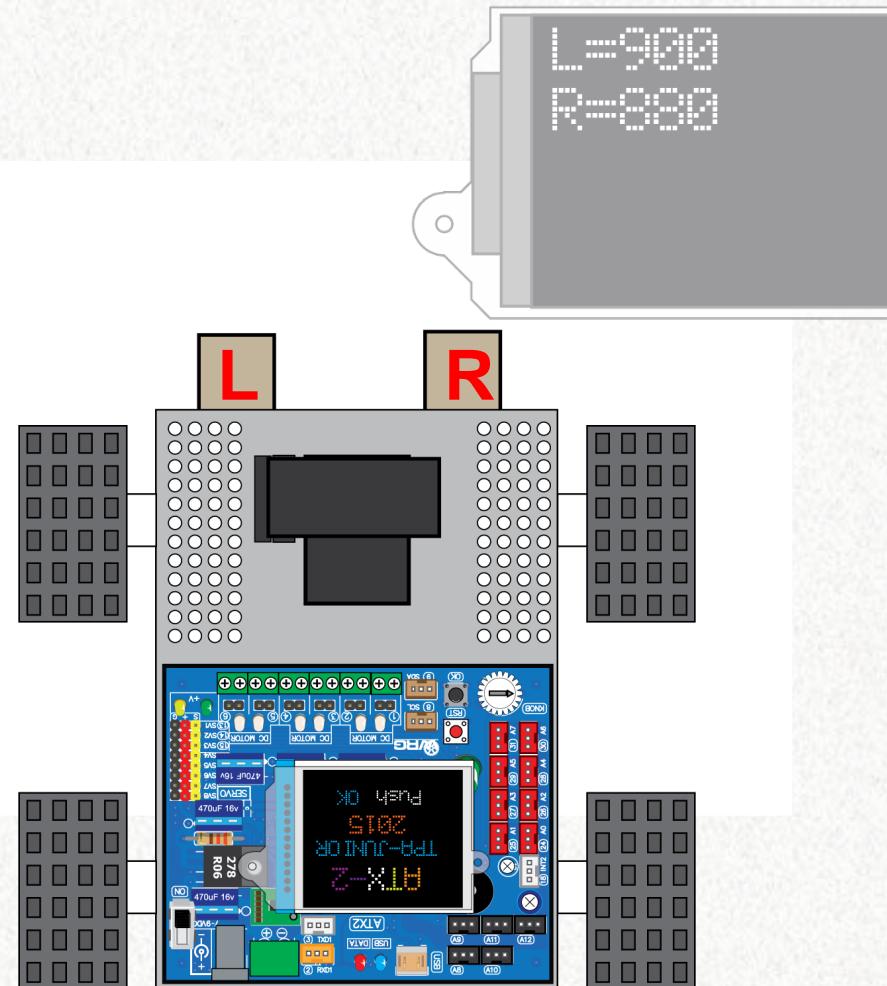
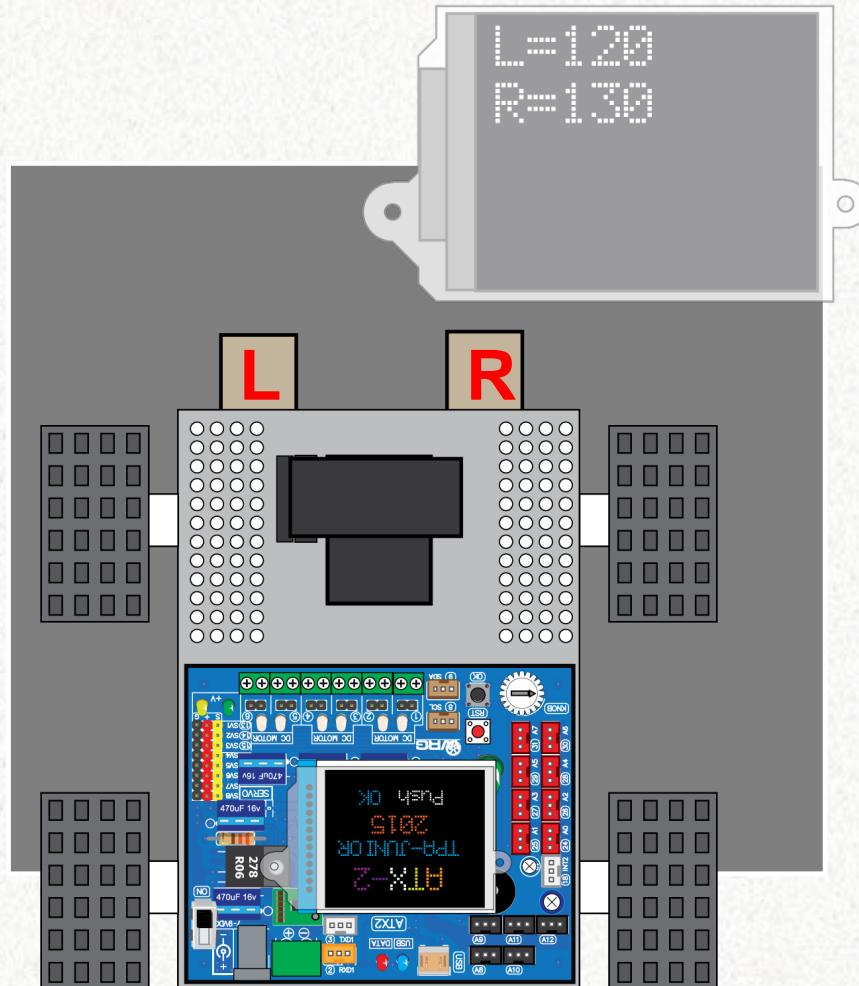
อ่านสีขาว

```
#include <ATX2.h>
void setup() {
    setTextSize(3);
    glcdMode(3);
}
void loop() {
    glcd(0,0,"L=%d      ",analog(5));
    glcd(1,0,"R=%d      ",analog(4));
}
```



อ่านสีดำ

ตรวจสอบผลการสะท้อนแสง ZX-03

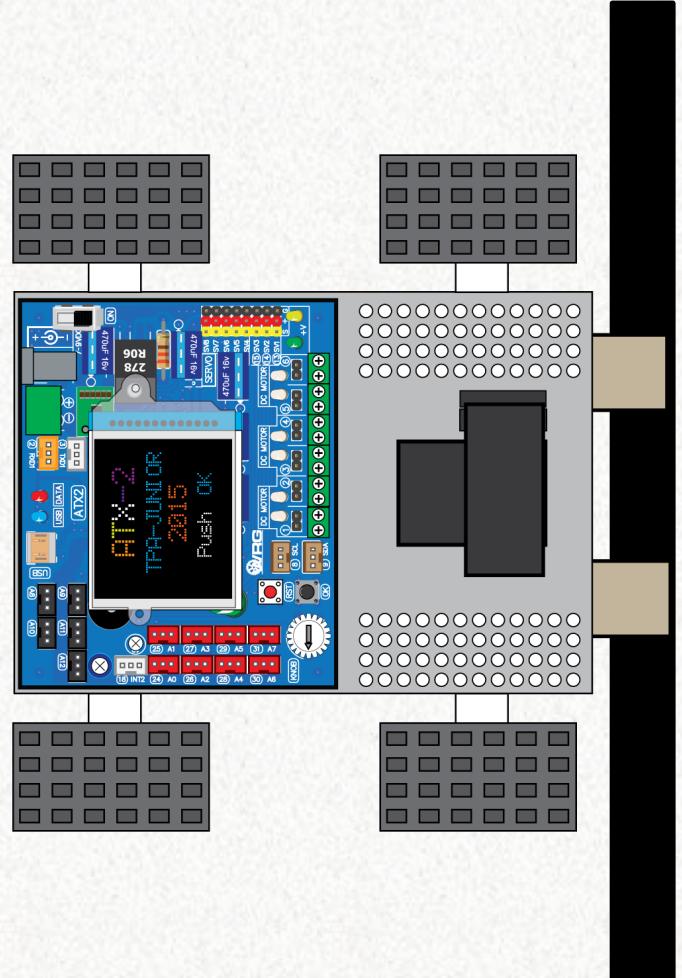


อ่านค่าสีดำ

อ่านค่าสีขาว

หุ่นยนต์เคลื่อนที่หยุดที่เส้นดำ

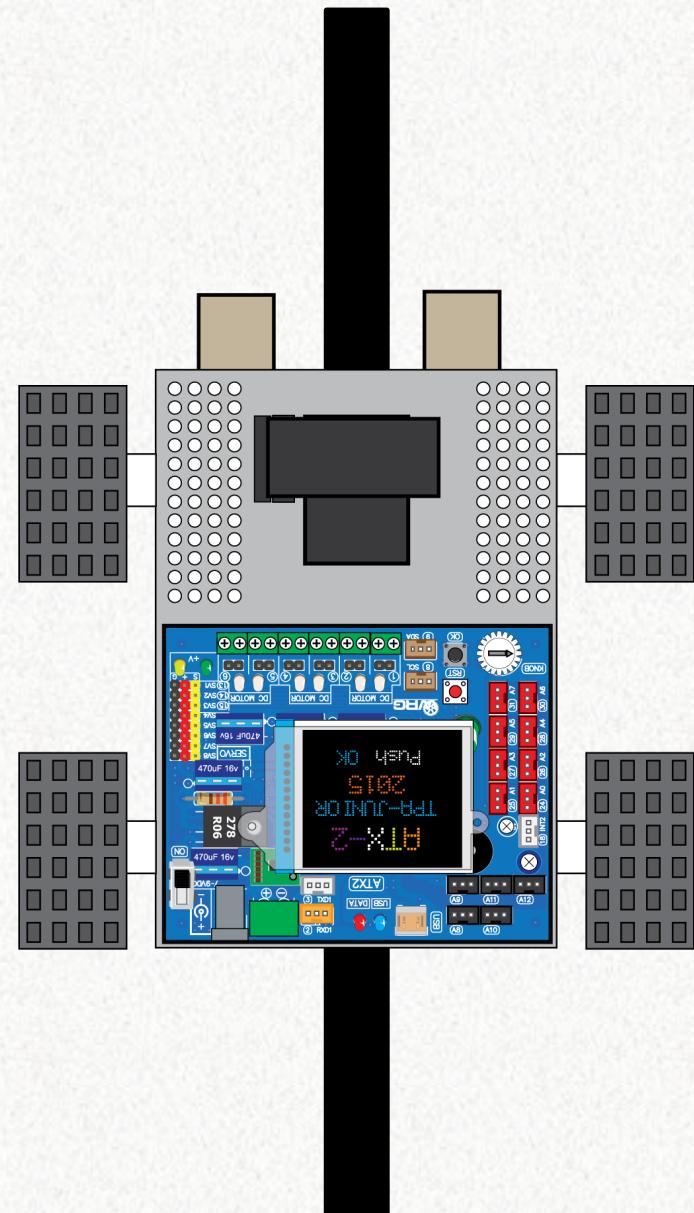
```
#include <ATX2.h>  
  
void setup() {  
    OK();  
    glcdClear();  
    FD(50);  
    while(analog(5)>510);  
    AO();  
    glcd(1,1,"Stop... ");  
}  
  
void loop() {}
```



รูปแบบการเคลื่อนที่ตามเส้นอย่างง่าย

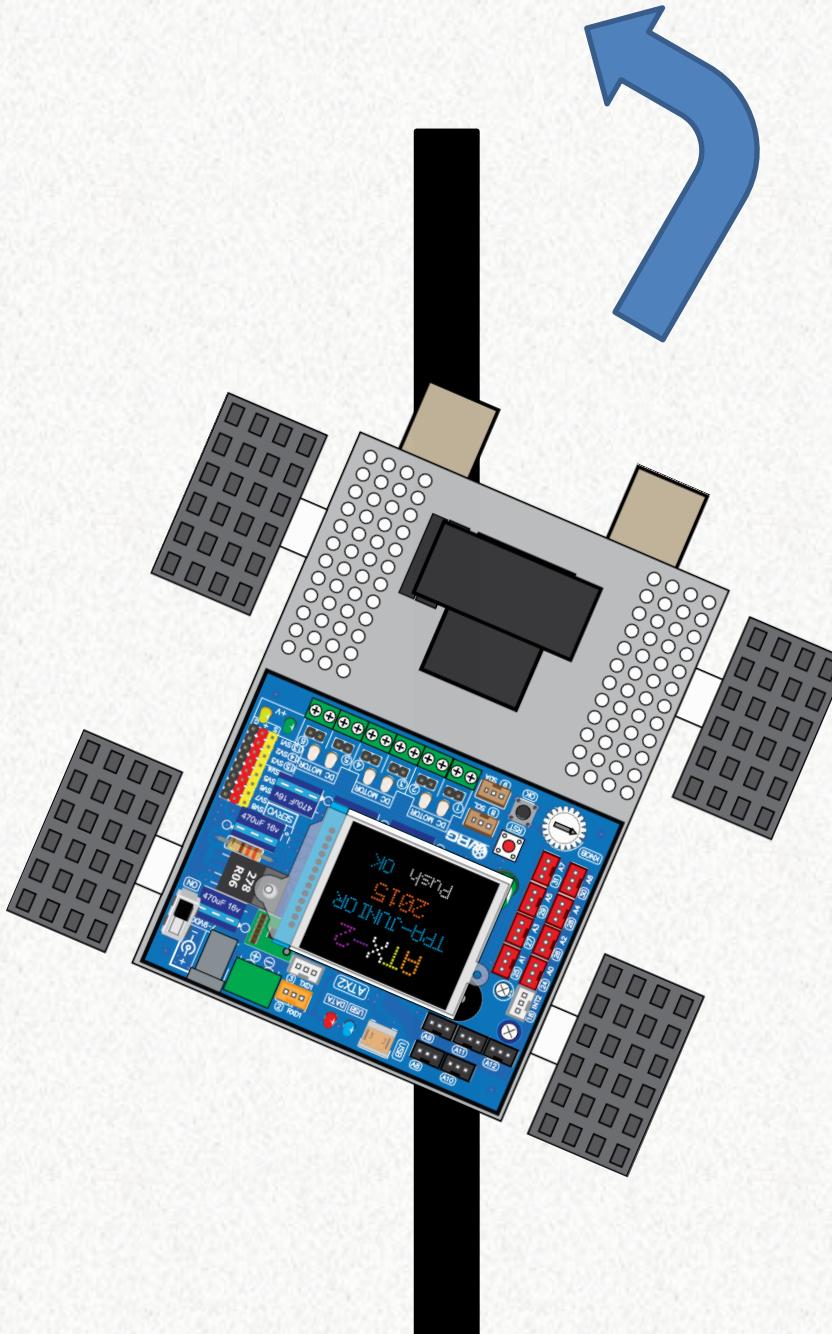
ด้านซ้ายและขวาเจอสีขาว

```
if (L>250 && R>250) {  
    FD (60);  
}
```



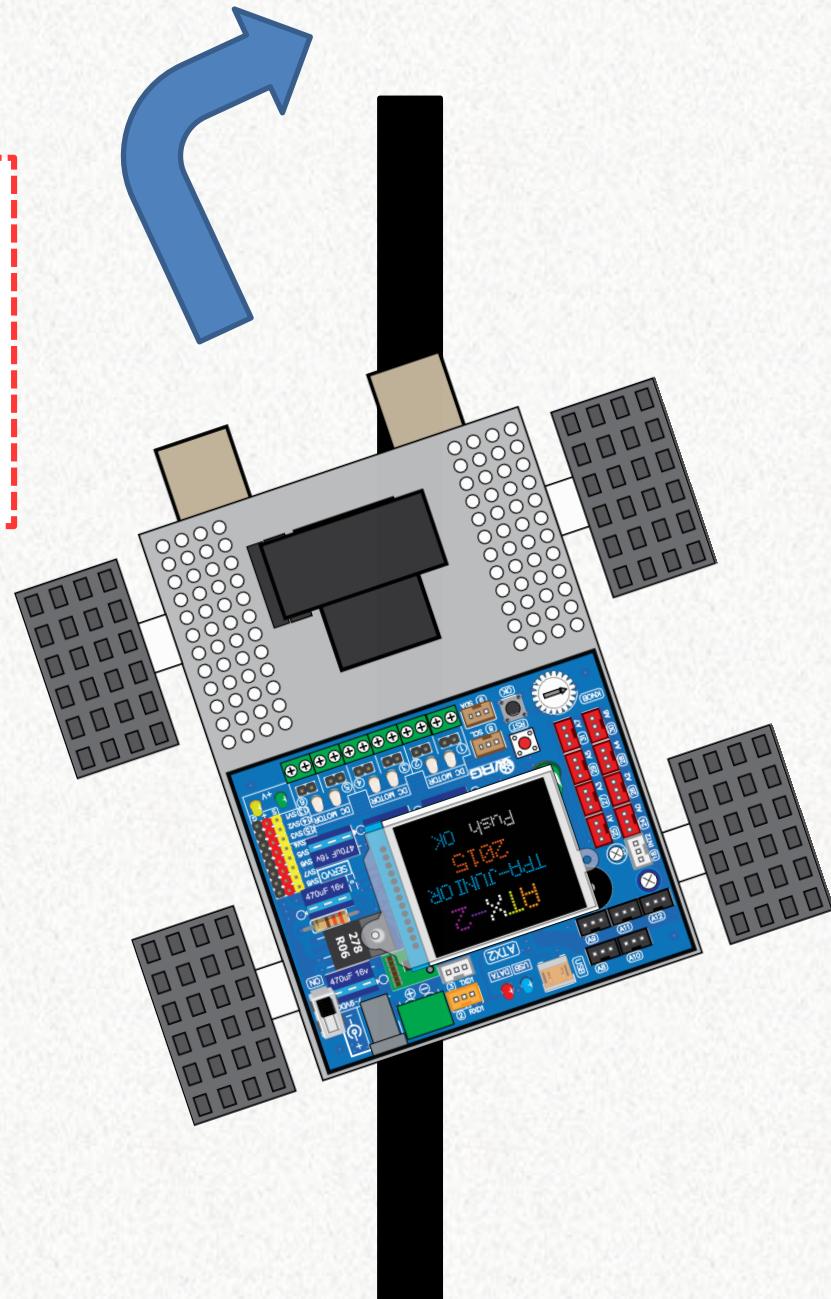
ด้านซ้ายเจอสีดำ

```
if (L<250 && R>250) {  
    SL(60);  
}
```



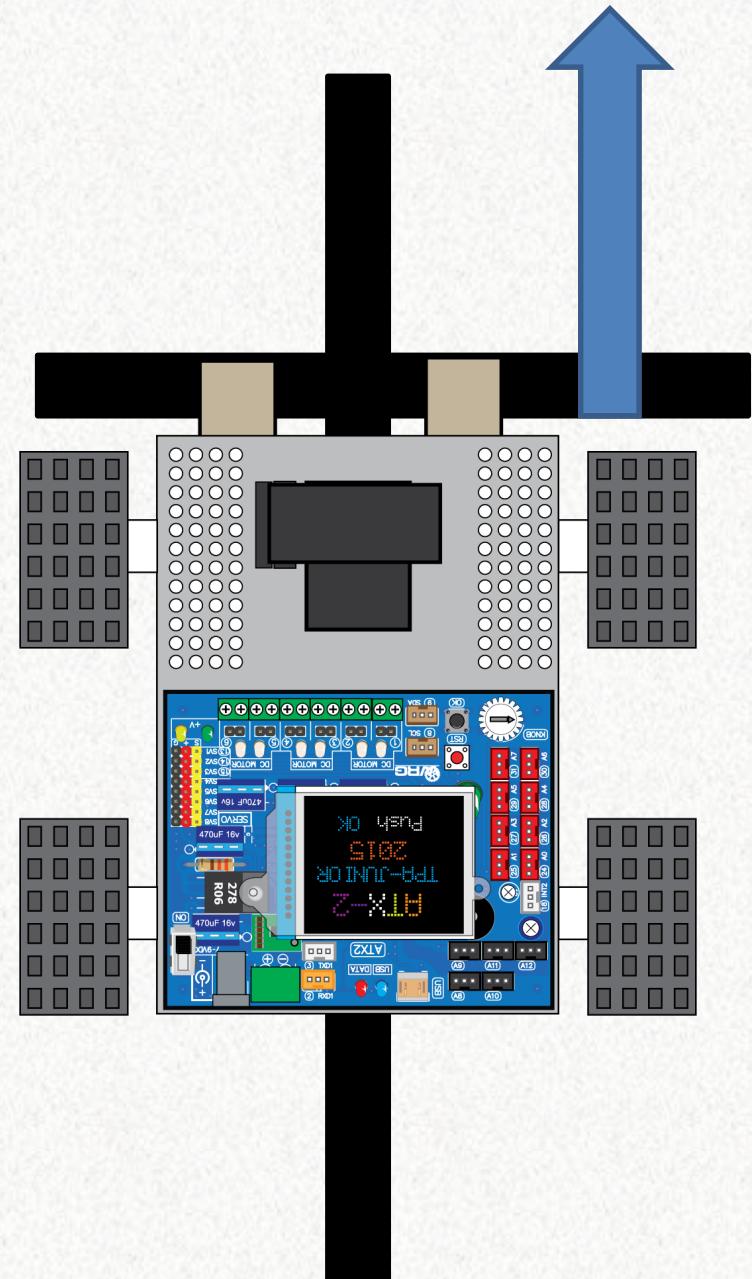
ด้านขวาเจอสีดำ

```
if (L>250 && R<250) {  
    SR(60);  
}
```

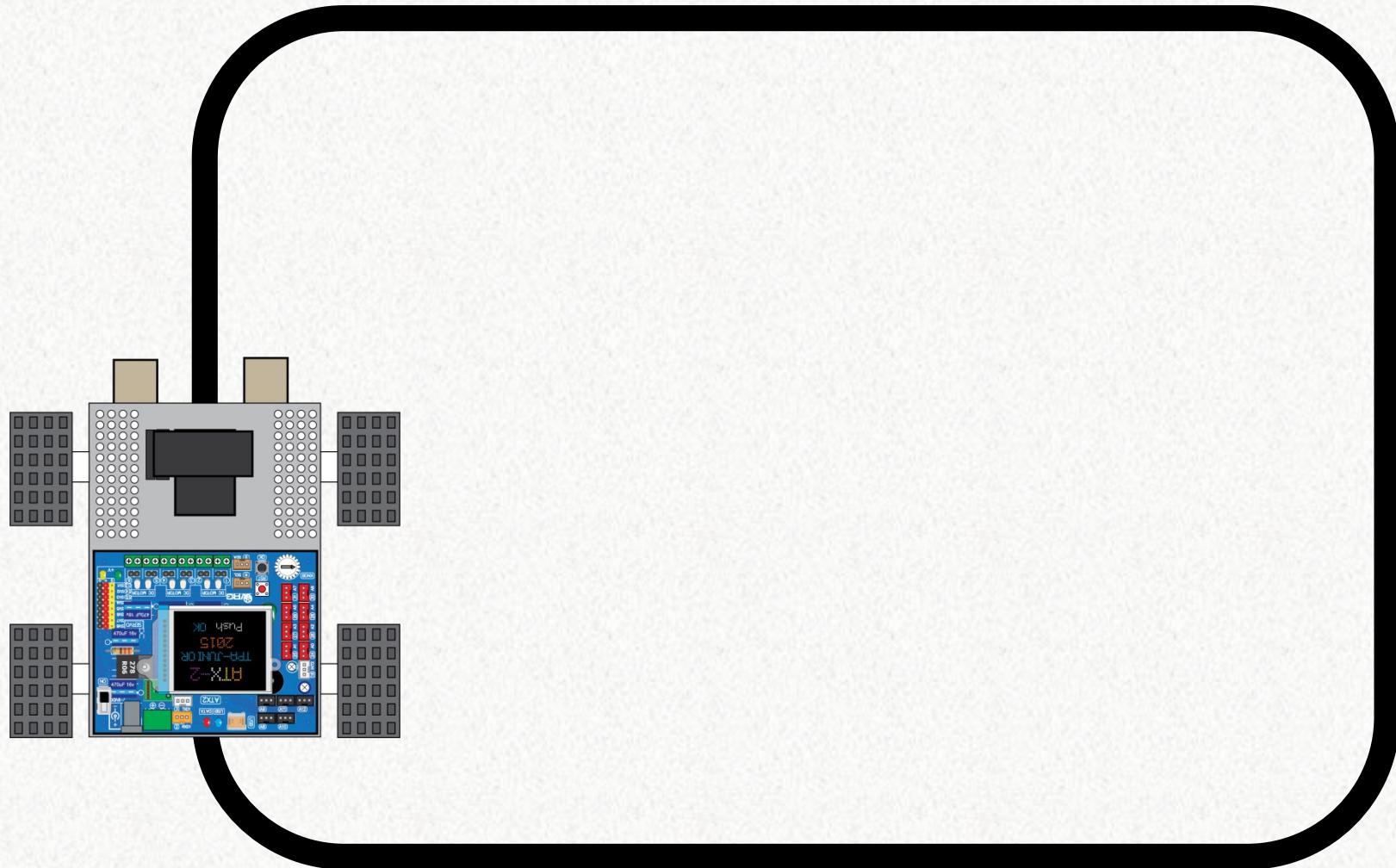


เจօສීදා ඝංදාන

```
if (L>250 && R<250) {  
    FD 60);  
    delay(200);  
}
```



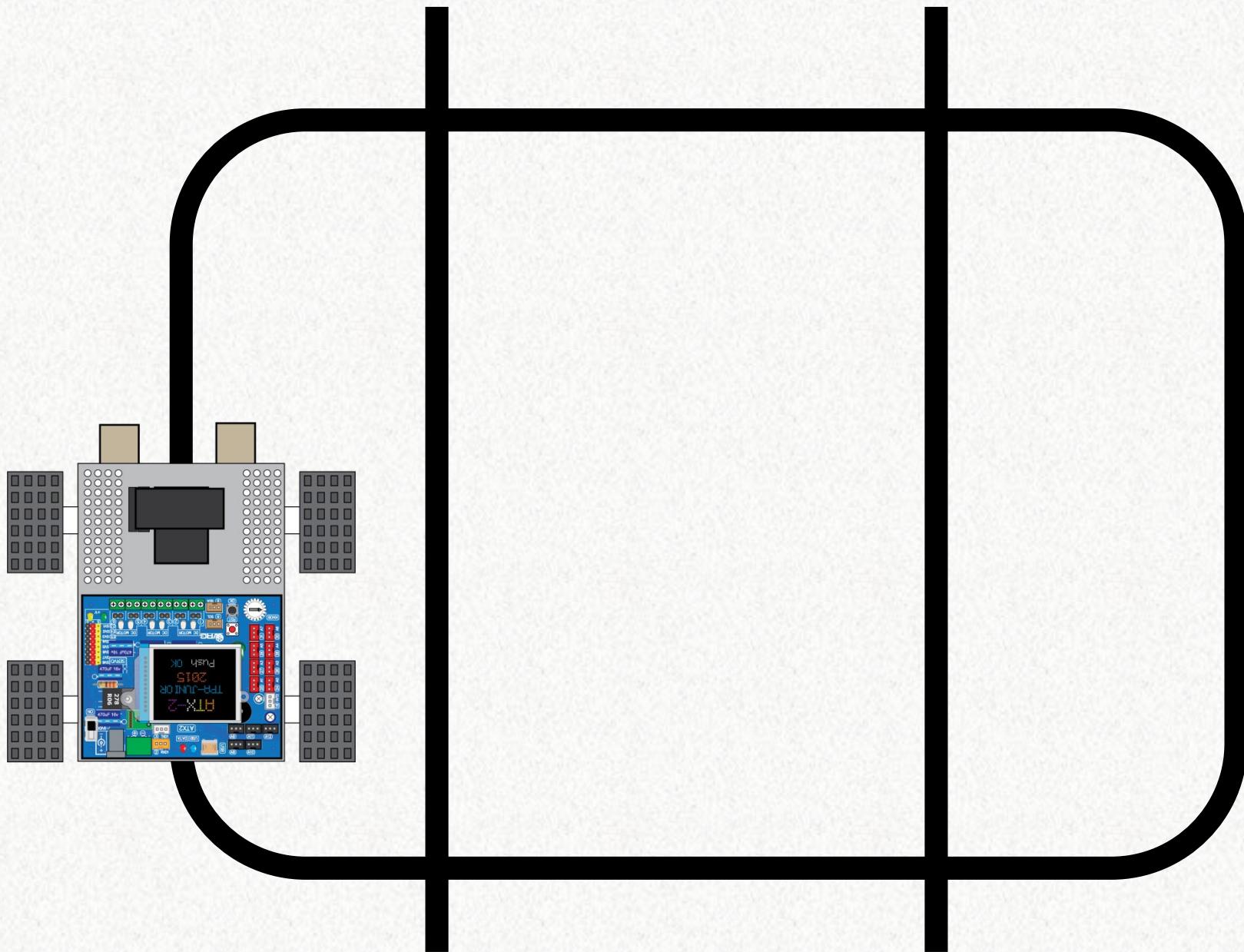
รูปแบบเคลื่อนที่ตามเส้นอย่างง่าย



โปรแกรมทดสอบ

```
#include <ATX2.h>
int L,R,RefL=510,RefR=505;
void setup() {OK(); }
void loop() {
    L=analog(5);
    R=analog(4);
    if(L>RefL&&R>RefR) { FD(40); }
    else if(L<RefL&&R>RefR) { SL(40); }
    else if(L>RefL&&R<RefR) { SR(40); }
}
```

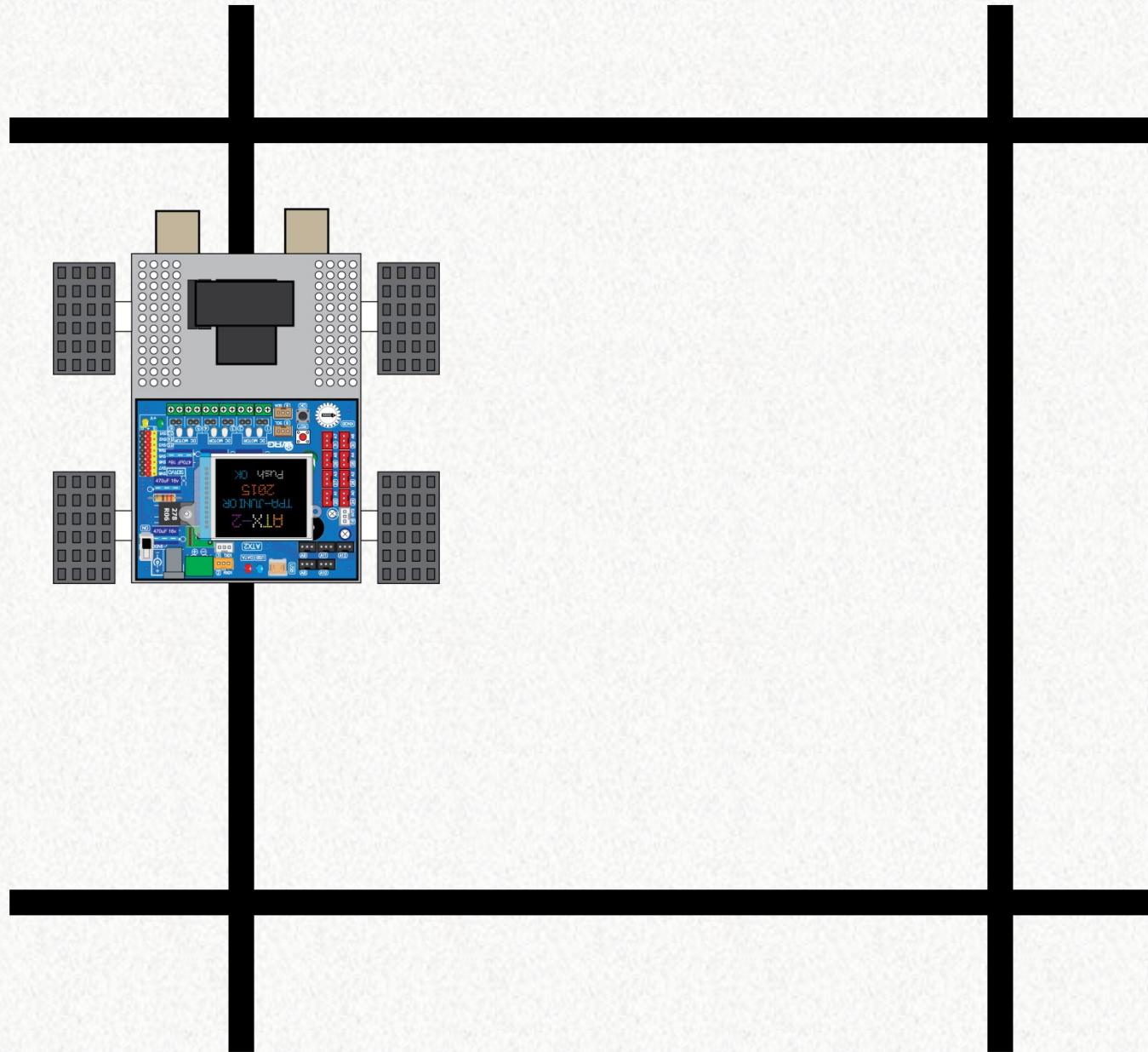
เจอเส้นตัดส่งเสียงออกลำโพง และเดินข้ามໄປ



โปรแกรมทดสอบ

```
#include <ATX2.h>
int L,R,RefL=510,RefR=505;
void setup() {OK(); }
void loop() {
    L=analog(5);
    R=analog(4);
    if(L>RefL&&R>RefR) { FD(40); }
    else if(L<RefL&&R>RefR) { SL(40); }
    else if(L>RefL&&R<RefR) { SR(40); }
    else if(L<RefL&&R<RefR) {
        FD(40); sound(500,200);
    }
}
```

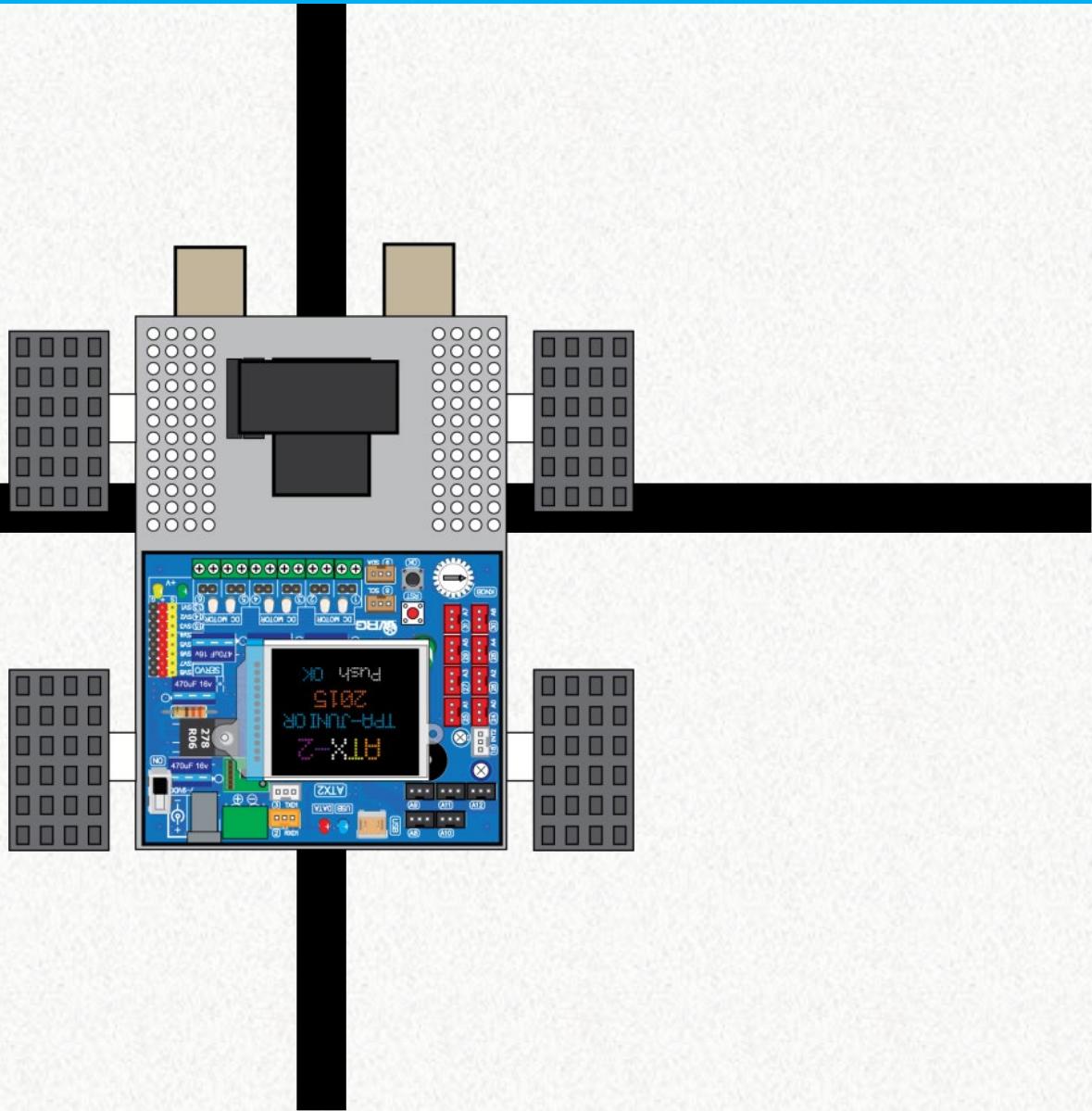
ເຈົ້າສັນຕິພົດເລື່ອງຂວາ



โปรแกรมทดสอบ

```
#include <ATX2.h>
int L,R,RefL=510,RefR=505;
void setup() {OK(); }
void loop() {
    L=analog(5);
    R=analog(4);
    if(L<RefL&&R<RefR) {
        FD(40);sound(500,200);
        SR(60);delay(400);
    }
    else if(L>RefL&&R>RefR) {FD(40); }
    else if(L<RefL&&R>RefR) {SL(40); }
    else if(L>RefL&&R<RefR) {SR(40); }
}
```

เทคนิคการเลี้ยวให้ได้ 90 องศาพอดี เมื่อเจอเส้นตัด



รูปแบบการสร้างฟังก์ชัน

ชื่อฟังก์ชัน ตัวแปรที่ส่งไปยังฟังก์ชัน

```
void R90(int x) {  
    ชุดคำสั่ง  
    y=x+2;  
    .  
}
```

ชุดคำสั่งในฟังก์ชัน

การใช้งานฟังก์ชัน

R90(200);

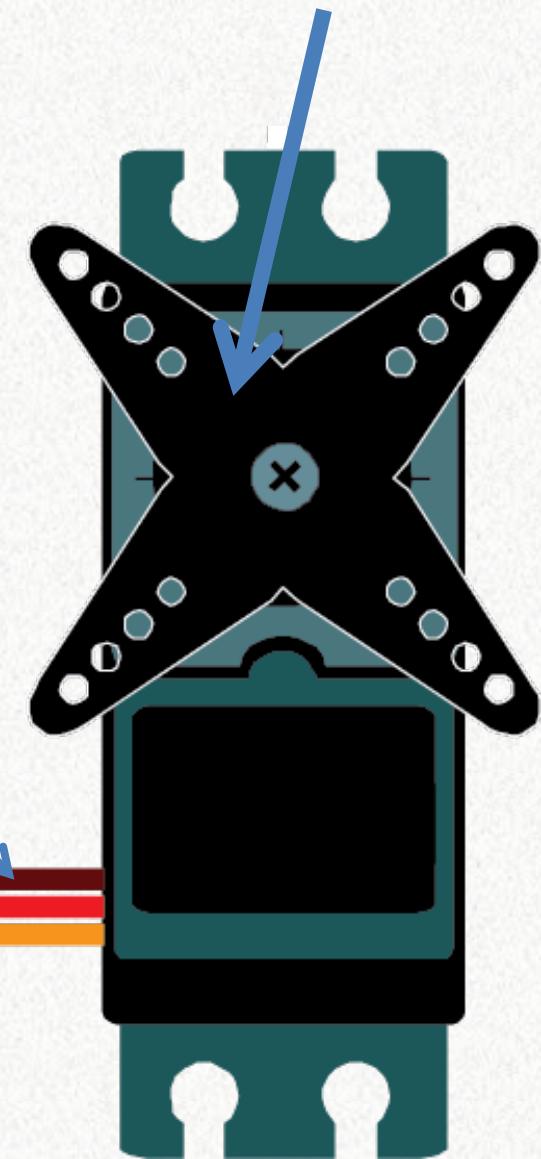
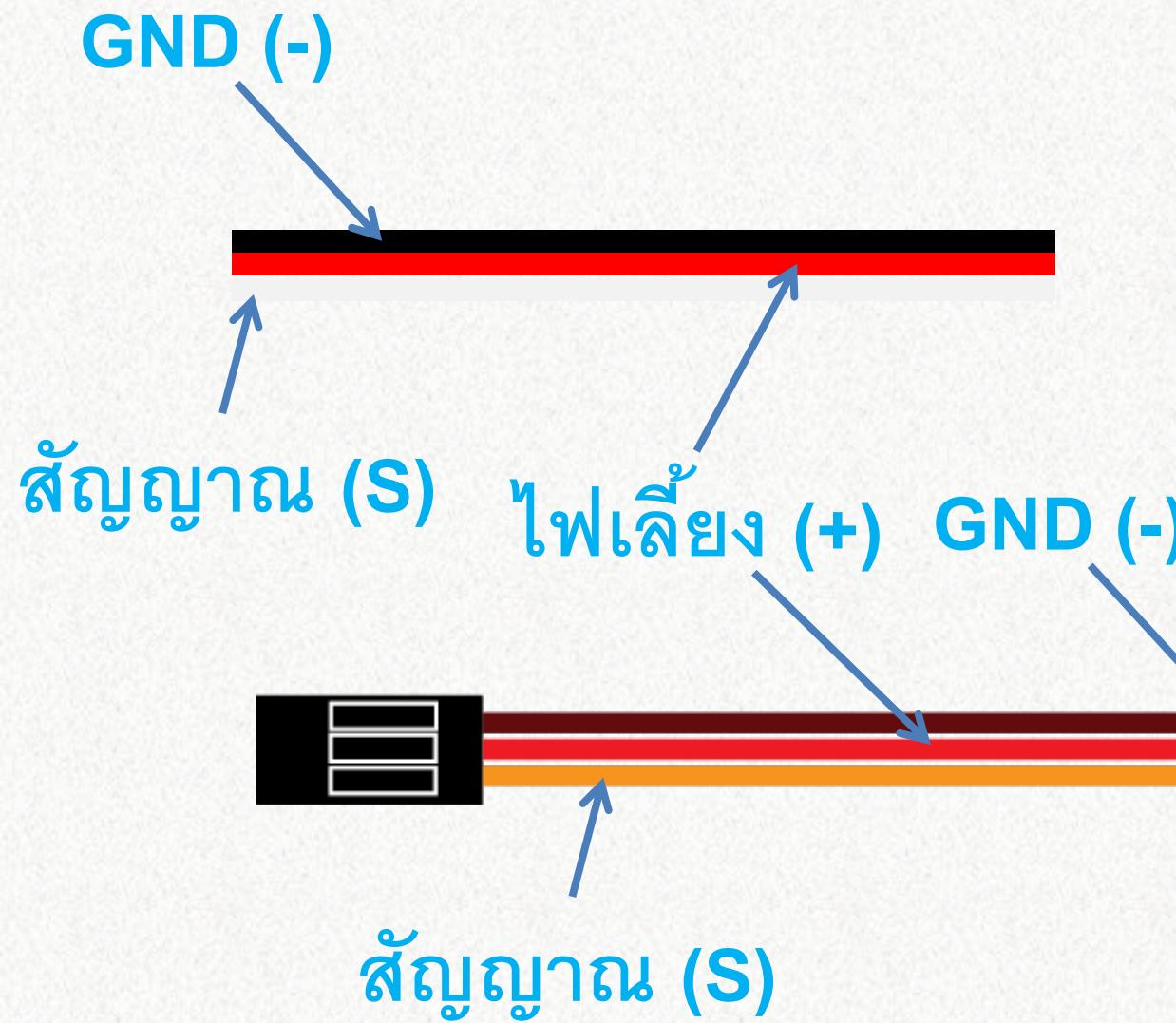
ฟังก์ชันเจอเส้นตัดเลี้ยวขวา/ซ้าย

```
void R90() {
    FD(40); sound(500,100);
    while(analog(5)>RefR) { SR(60); }
    while(analog(5)<RefR) { SR(60); }
}

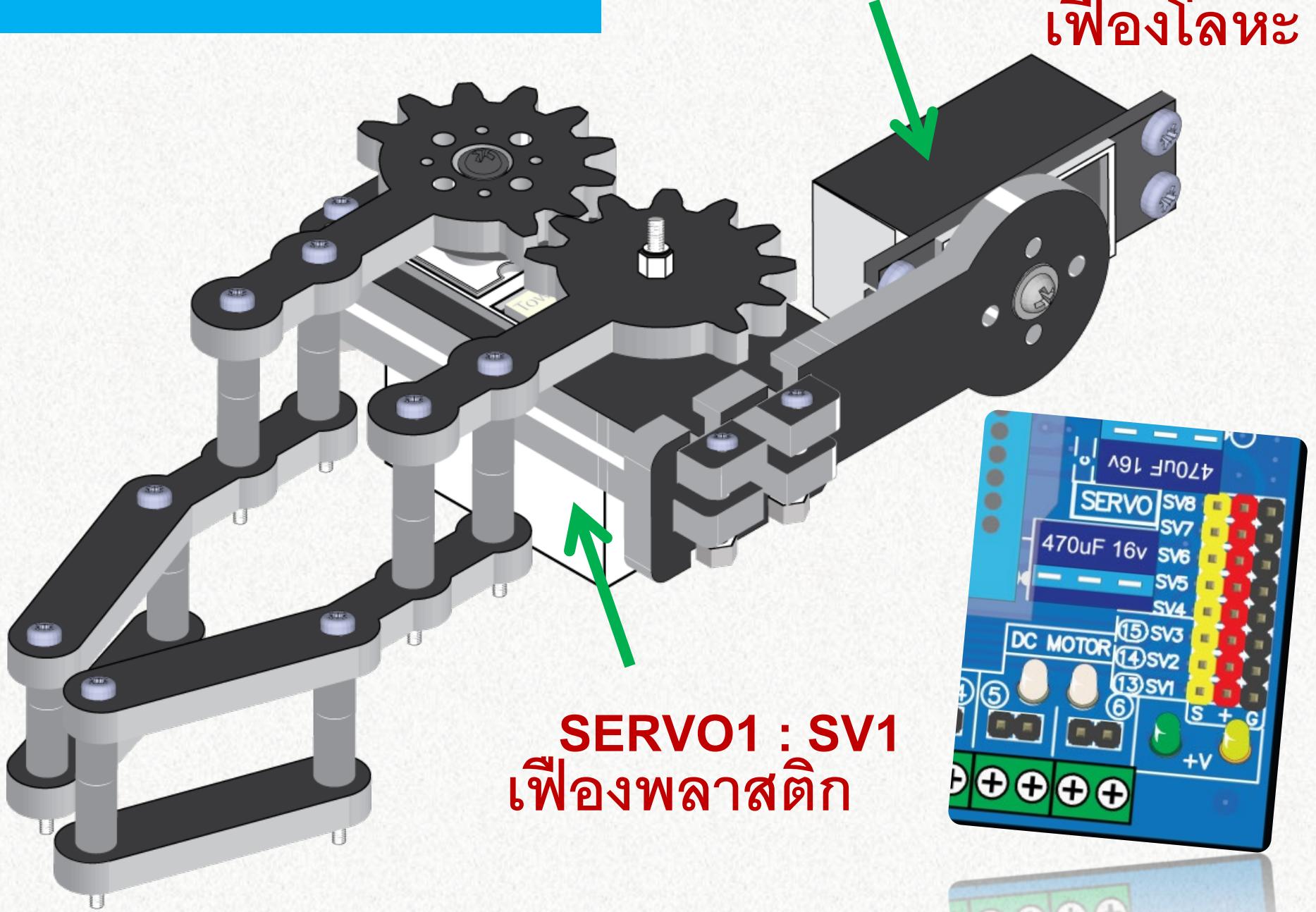
void L90() {
    FD(40); sound(500,100);
    while(analog(4)>RefL) { SL(60); }
    while(analog(4)<RefL) { SL(60); }
}
```

เซอร์โวมอเตอร์มาตรฐาน

แกนหมุน 180 องศา



SM Gripper 2015



ฟังก์ชันขับเซอร์โวมอเตอร์

servo (CH, POS);

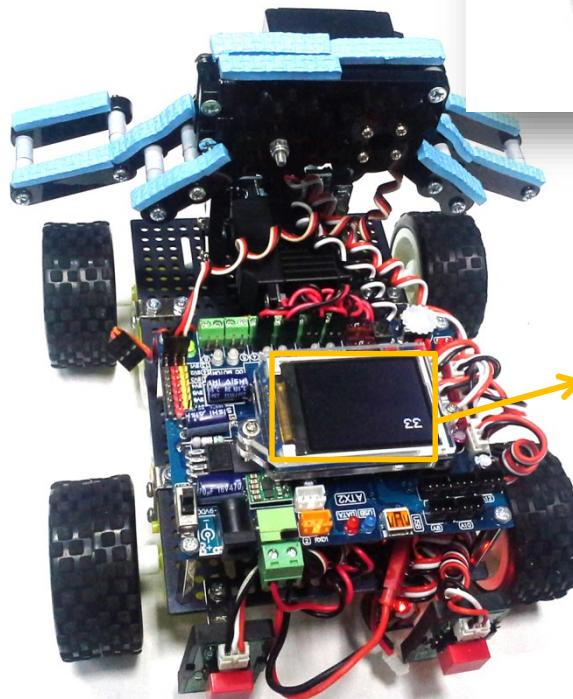
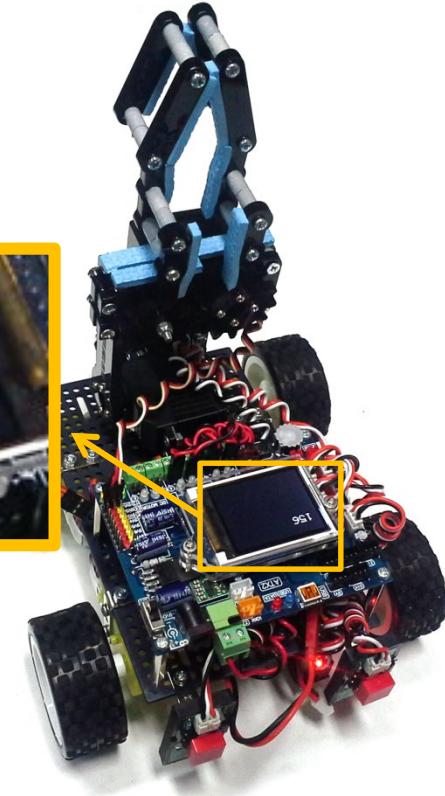
CH ช่องที่ใช้ขับ = 1 ถึง 8

POS ตำแหน่งองศาเซอร์โว = 0-180 , -1

ค่า -1 หมายถึงหยุดจ่ายสัญญาณให้เซอร์โว
เซอร์โวจะไม่ล็อกแกน

โปรแกรมทดสอบ SERVO1

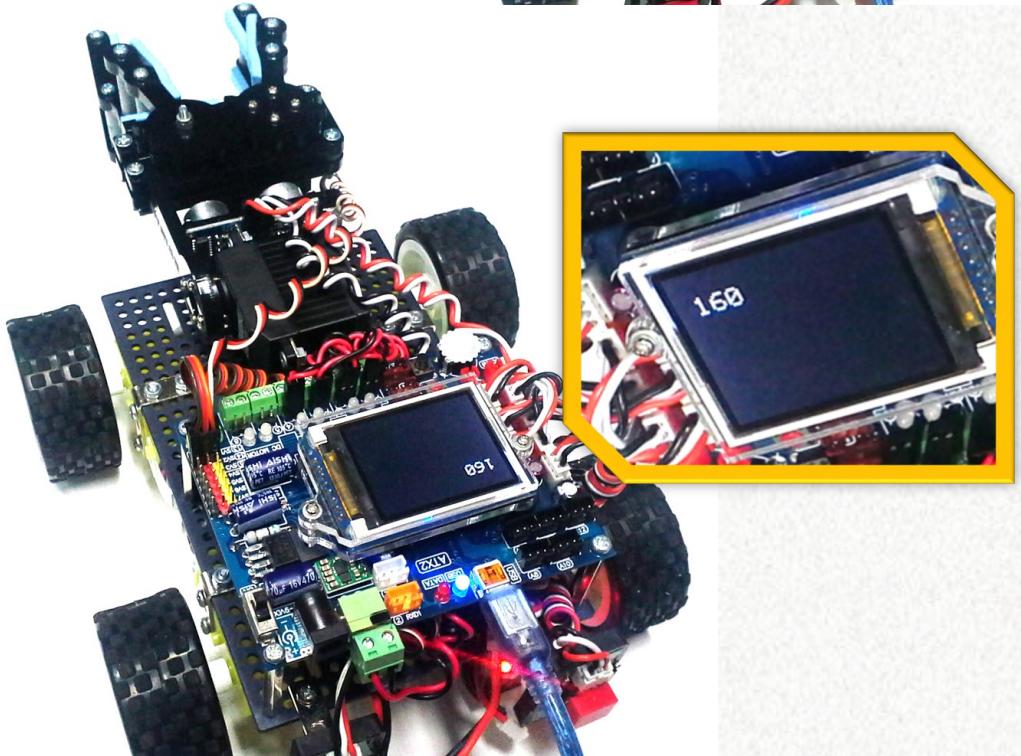
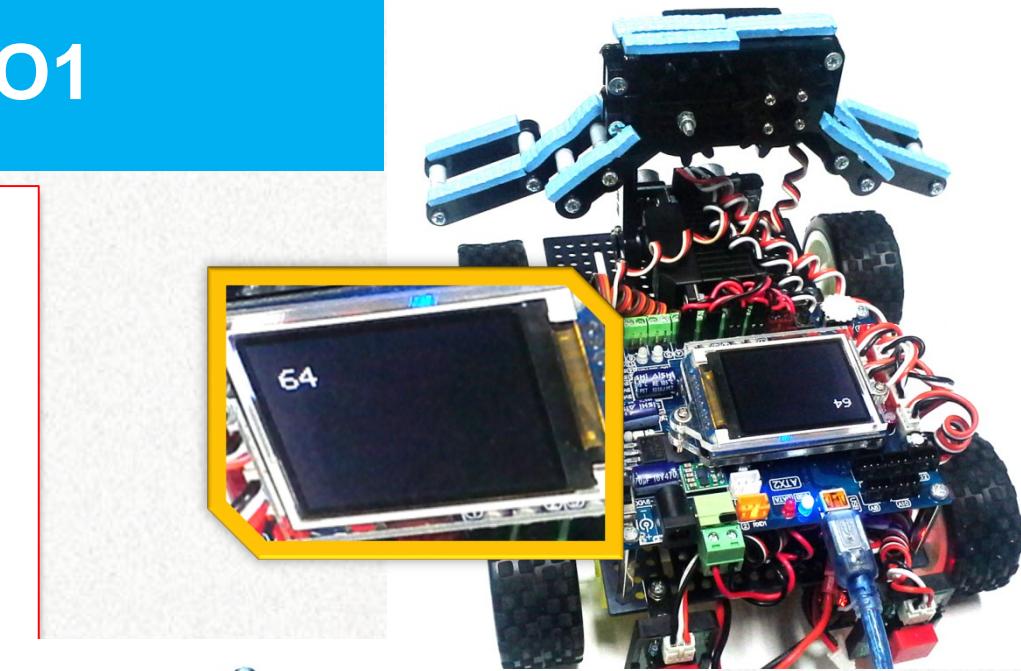
```
#include <ATX2.h>
int x;
void setup() {
    OK(); glcdClear();
}
void loop() {
    x=knob(180);
    servo(1,x);
    glcd(1,1,"%d",x);
}
```



โปรแกรมทดสอบ SERVO1

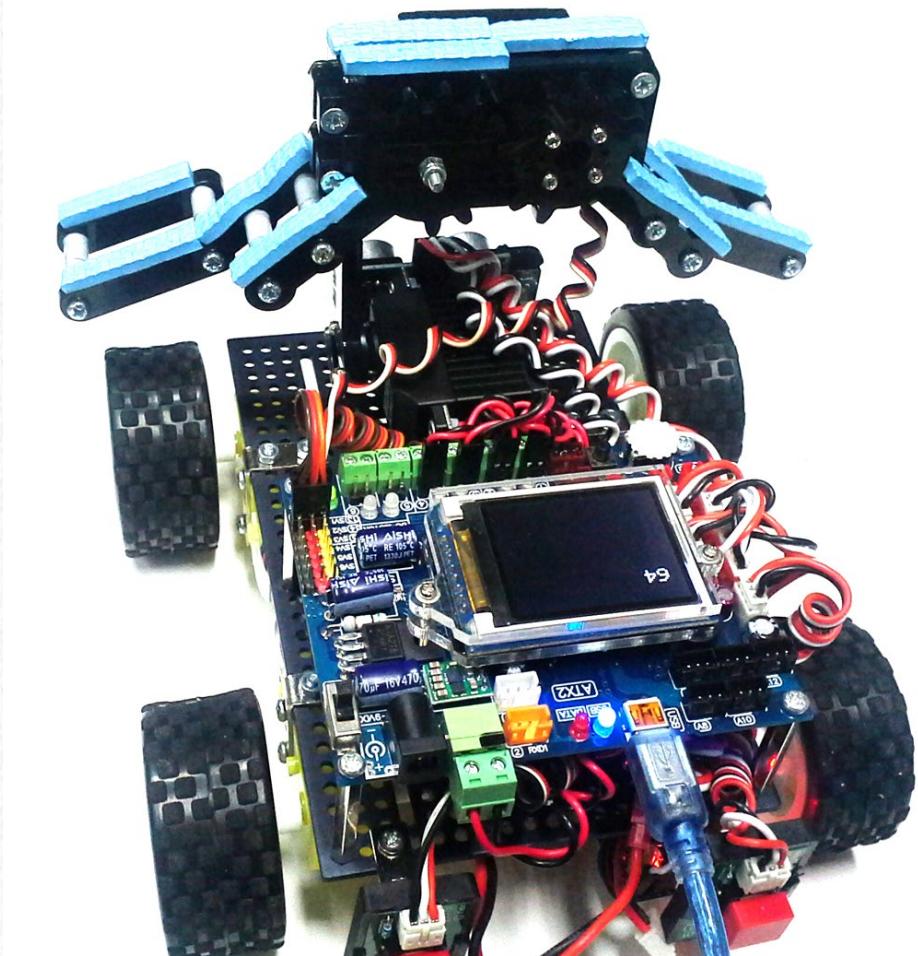
```
#include <ATX2.h>
int x;
void setup() {
    OK();glcdClear();
}
void loop() {
x=knob(180);
servo(2,x);
glcd(1,1,"%d    ",x);
}
```

หมุน knob ทดสอบ

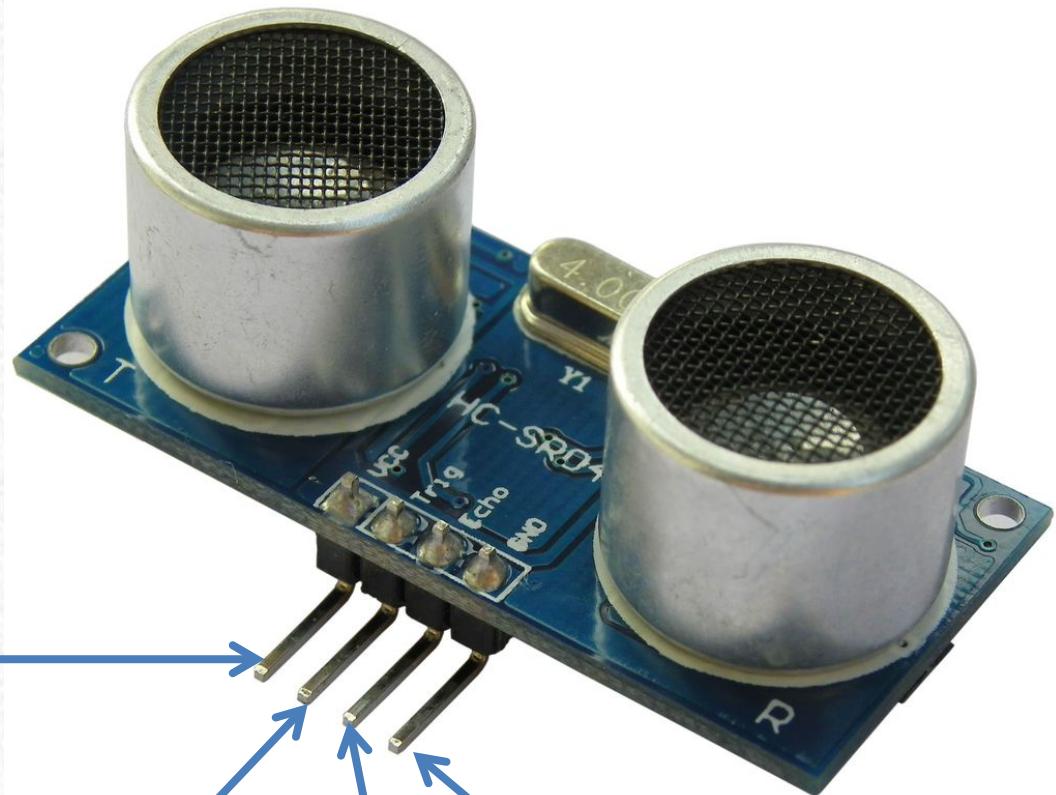


สร้างฟังก์ชัน sHome()

เพื่อให้เซอร์โวอยู่ในตำแหน่ง home



อัลตร้าโซนิค HC-SR04



ไฟเลี้ยง (+)

Trigger(24)

Echo(25)

GND (-)

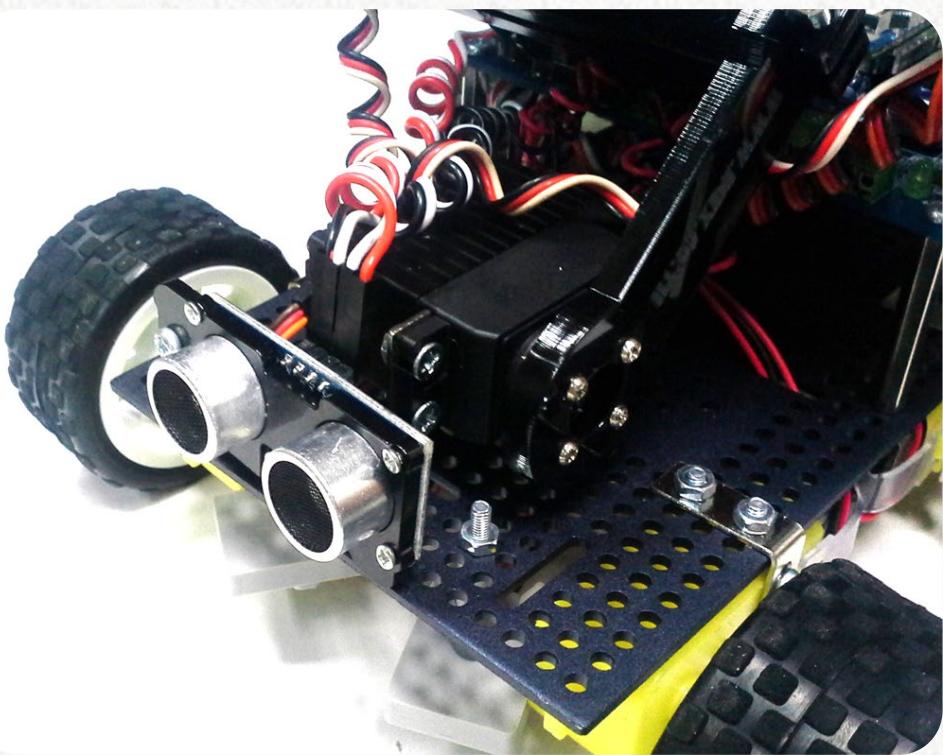
ฟังก์ชันอ่านค่าจาก HC-SR04

sonar () ;

คืนค่าระยะทางเป็นเซนติเมตร

_sonarTime () ;

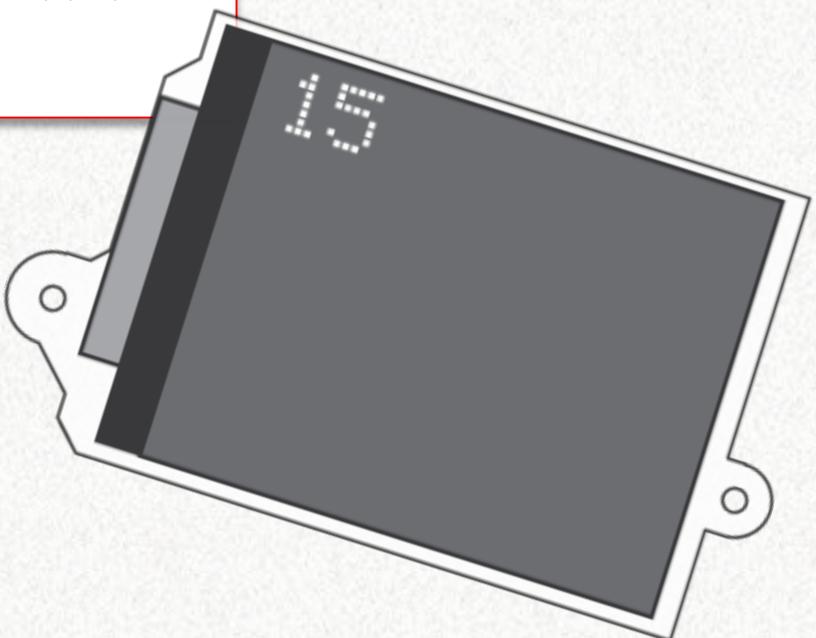
คืนค่าระยะทางเป็นค่าดิบ



ตัวอย่างการใช้งานฟังก์ชัน Sonar

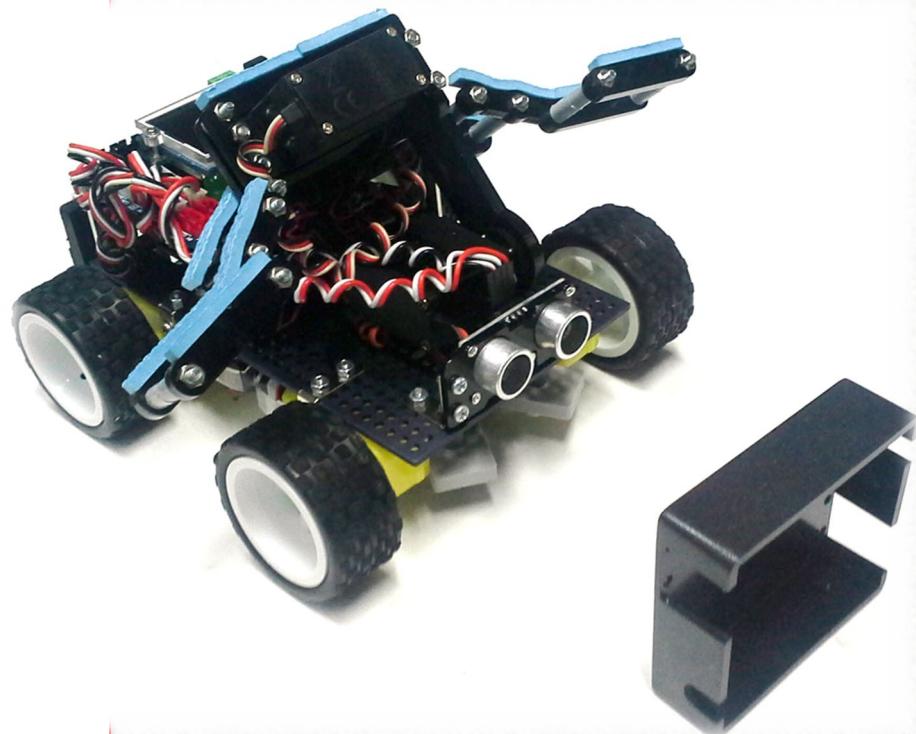
```
#include <ATX2.h>
void setup() {
    OK(); glcdClear();
}
void loop() {
    glcd(0,0,"%d", sonar());
}
```

แสดงค่าเป็นเซนติเมตร



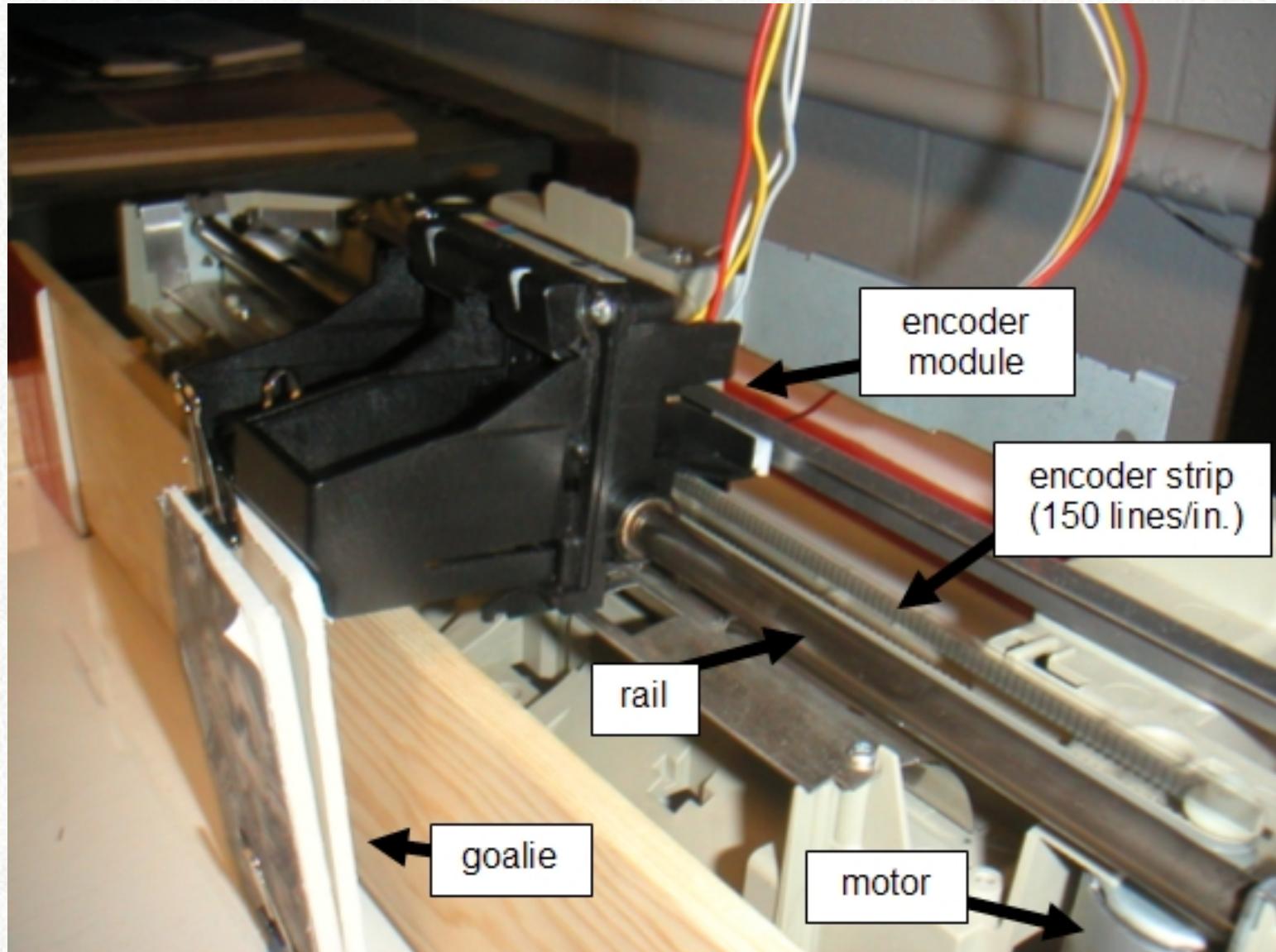
ตัวอย่างการใช้งานฟังก์ชัน Sonar

```
#include <ATX2.h>
int x;
void setup() {
    OK(); glcdClear();
    FD(100);
    while (sonar()>8);
    BK(100); delay(50); AO();
}
void loop() {
    sw_OK_press();
    FD(100);
    while (sonar()>8);
    BK(100); delay(50); AO();
}
```



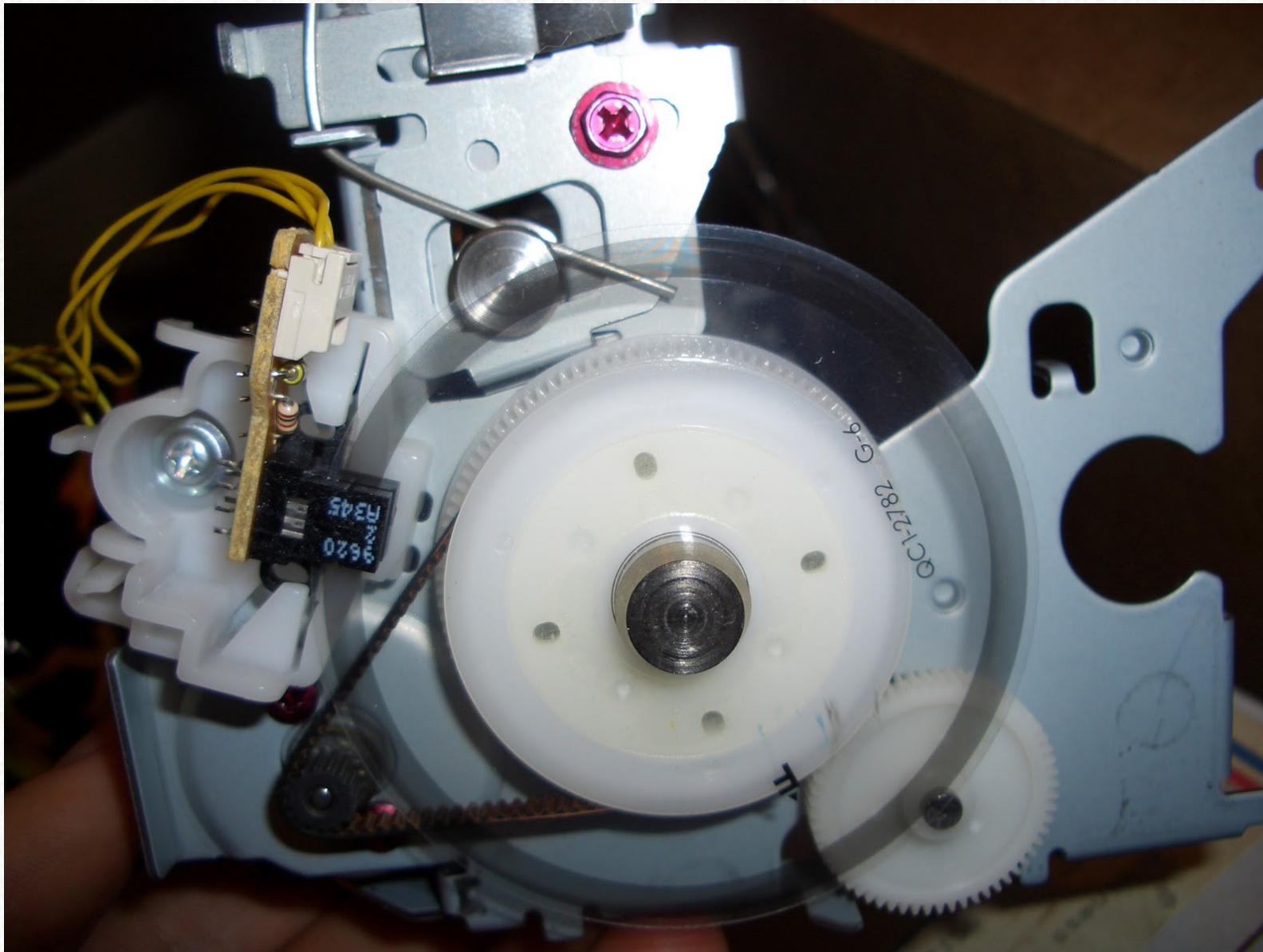
Encoder

เครื่องพิมพ์ Inkjet

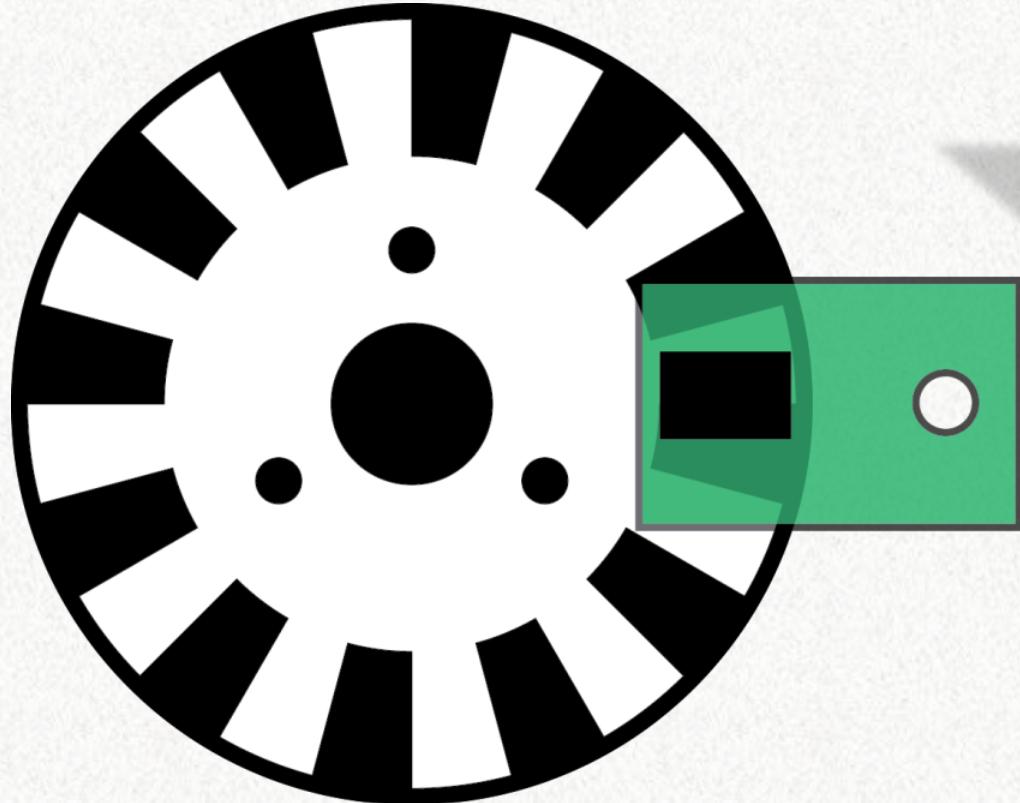


Encoder

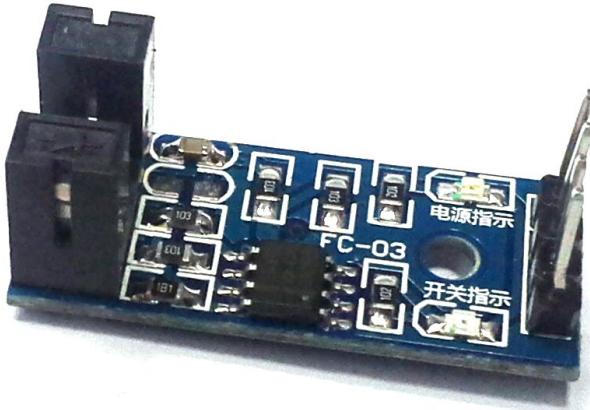
เครื่องพิมพ์ Inkjet



Encoder อย่างง่ายจาก ZX-03Q



Encoder แบบก้ามปูสำหรับ RQ-Bot



Encoder ก้ามปู

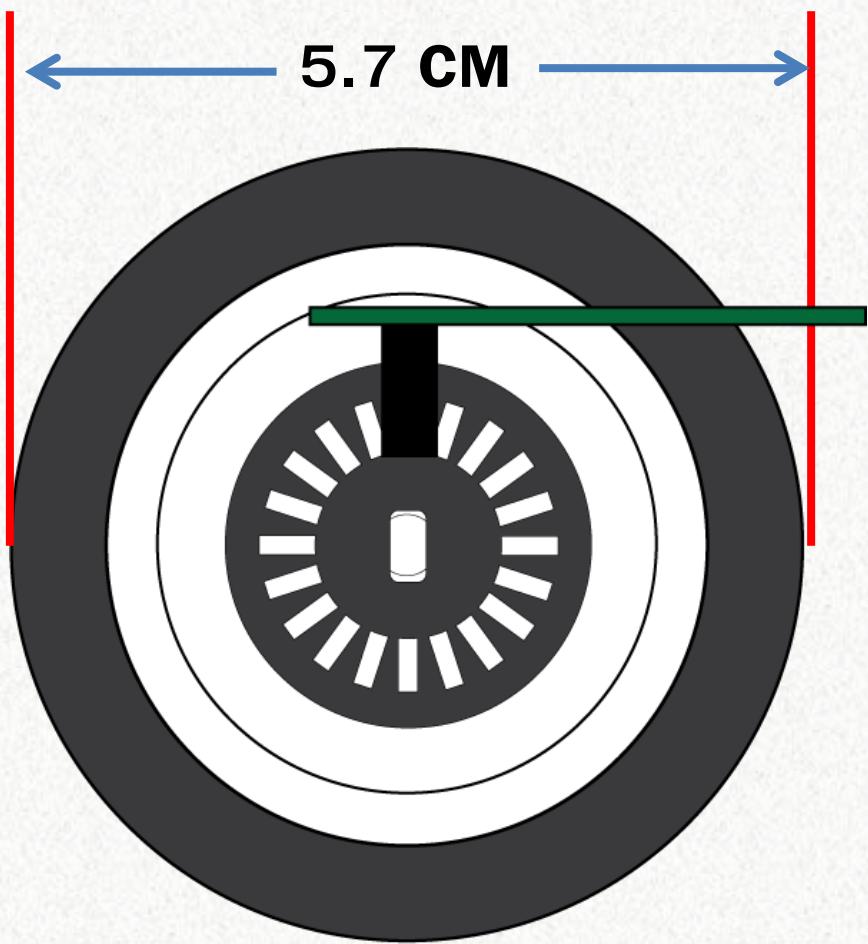


จาน Encoder 20 ร่อง



ติดตั้งที่ล้อหลังขวา
ต่อสายเข้าที่ขา 18/INT2

พังก์ชันเจօเส็นต์ดเลี้ยวขวา/ซ้าย



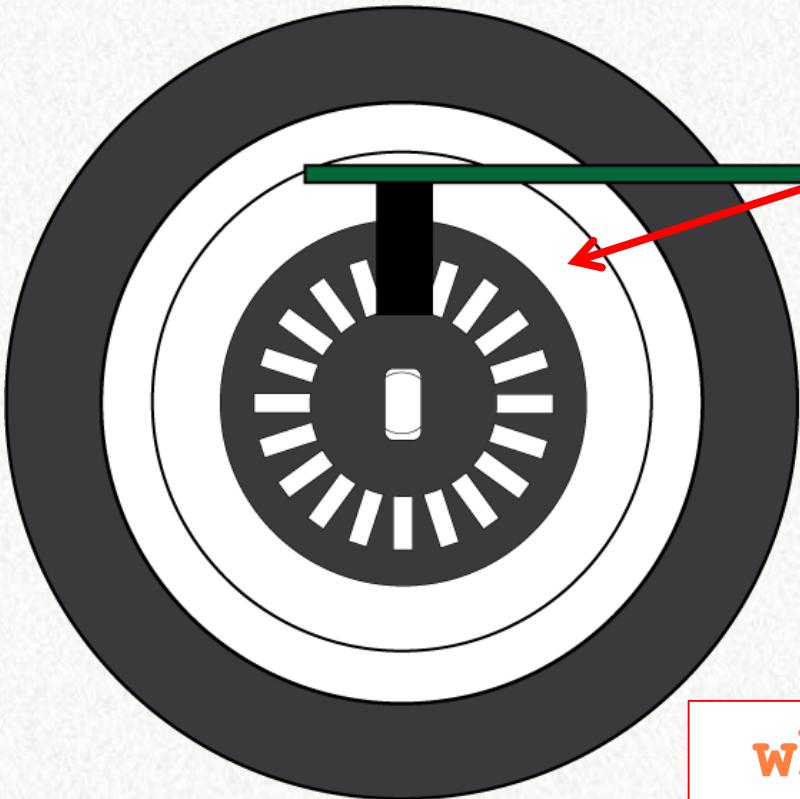
จำนวนช่องเท่ากับ **20** ช่อง
เส้นผ่านศูนย์กลาง **5.7 CM**

$$\text{เส้นรอบวง} = \pi D$$

$$3.1415 \times 5.7 = 17.75$$

$$\text{เส้นรอบวง} = 17.75 \text{ CM}$$

แนวคิดการเขียนโปรแกรมอย่างง่าย



อ่านเป็นขา 1 ครั้ง ดำเนิน 1 ครั้ง
นับเป็น 1 ค่า

```
while(x<20) {  
    while(in(18));  
    while(!in(18));  
    x++;  
}
```

เรียกใช้งานผ่านไลบรารี ATX2_enc.h

encCnt

1. ตัวแปร encCnt นี้จะเพิ่มค่าอัตโนมัติ (เมื่อเกิดการเปลี่ยนแปลงที่ Encoder)
2. เพิ่มทั้งกรณีเจอช่องว่างและไม่เจอช่องว่าง
3. ถ้าต้องการนับค่าใหม่ ให้ใช้ **encCnt =0;**
4. ตรวจสอบค่าได้ง่าย ๆ เช่น
while (encCnt<50) ;

ตัวอย่างการใช้งาน

```
#include <ATX2.h>
void setup() {
    OK(); glcdClear();
}
void loop() {
    glcd(0,0,"%d
",encCnt);
}
```

ทดลองหมุนล้อดูค่าที่เปลี่ยนแปลง

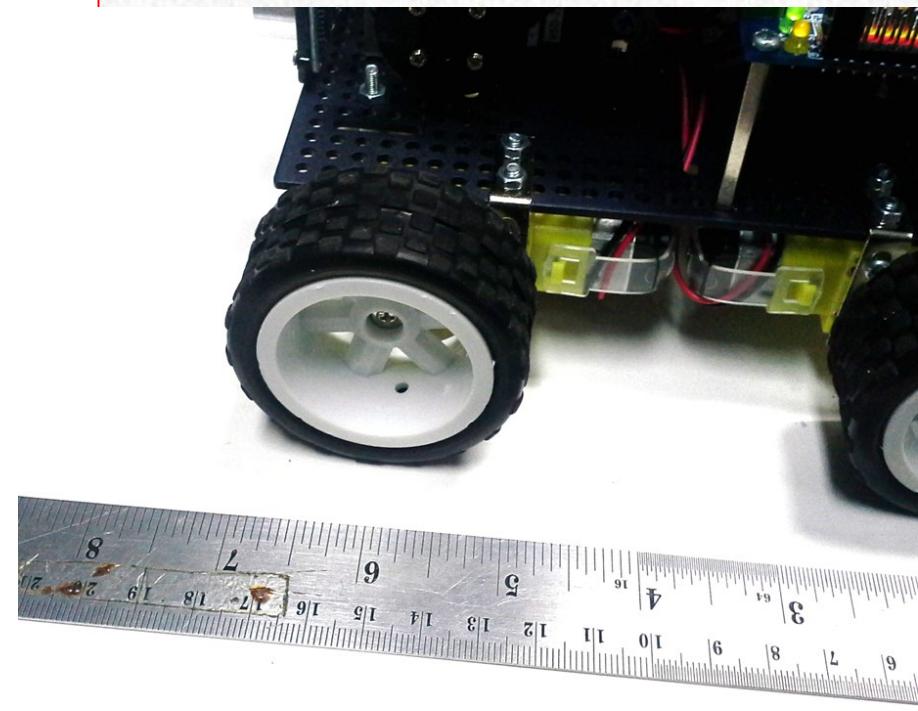


ตัวอย่างการใช้งาน

หยุดกระแทกหนาเมื่อหมุนครบรอบ

```
#include <ATX2.h>
void setup() {
    OK(); lcdClear();
    FD(100);
    while(encCnt<40);
    encCnt=0;
    BK(100); delay(50); AO();
}
void loop() {
    sw_OK_press();
    FD(100);
    while(encCnt<40);
    encCnt=0;
    BK(100); delay(50); AO();
}
```

17.75 CM



เขียนเมนูอย่างง่าย

```
#include <ATX2.h>
int k;
void setup() {
OK();glcdClear(); }
void loop() {
k=knob(3);
glcd(0,0,"Prog %d ",k+1);
if(sw_OK()){
  if(k==0){P1();}
  else if(k==1){P2();}
  else if(k==2){P3();}
  else if(k==3){P4();}
}}
```

```
void P1(){
while(1){
  glcd(1,1,"L=%d
",analog(0));
  glcd(3,1,"R=%d
",analog(1));
}
void P2(){
while(1){beep(16);}}
void P3(){
while(1){fd(100);}}
void P4(){
while(1){bk(100);}}
```