Cytron line following using the LSA08 Auto- Callibrating line following sensor

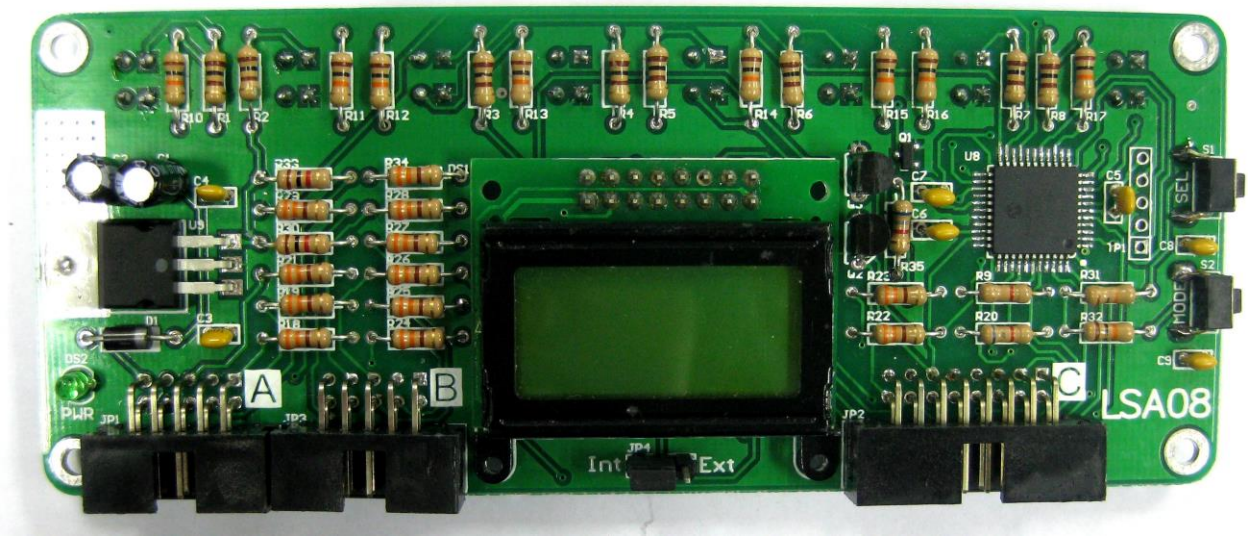
LSA08 (Advance Line Following Sensor Bar) consist of 8 IR transmitter and IR receiver pairs. LSA08 is typically used for embedded system or robots for line following task.
LSA08 can detect any color of line which has brightness different with the background. The IR transmitters on LSA08 are pulsed to allow the transmitter to off at certain idle period of sensor. This minimizes the current consumption of LSA08 to at least half of the current consumption compared to a normal unregulated IR line sensor. Power polarity protection is available on LSA08 in case the user accidentally applies a reverse voltage. LSA08 has several different output modes, for the convenience of use for any system. Namely, the digital output port (8 parallel output line), the serial communication port (UART) and the analog output port.

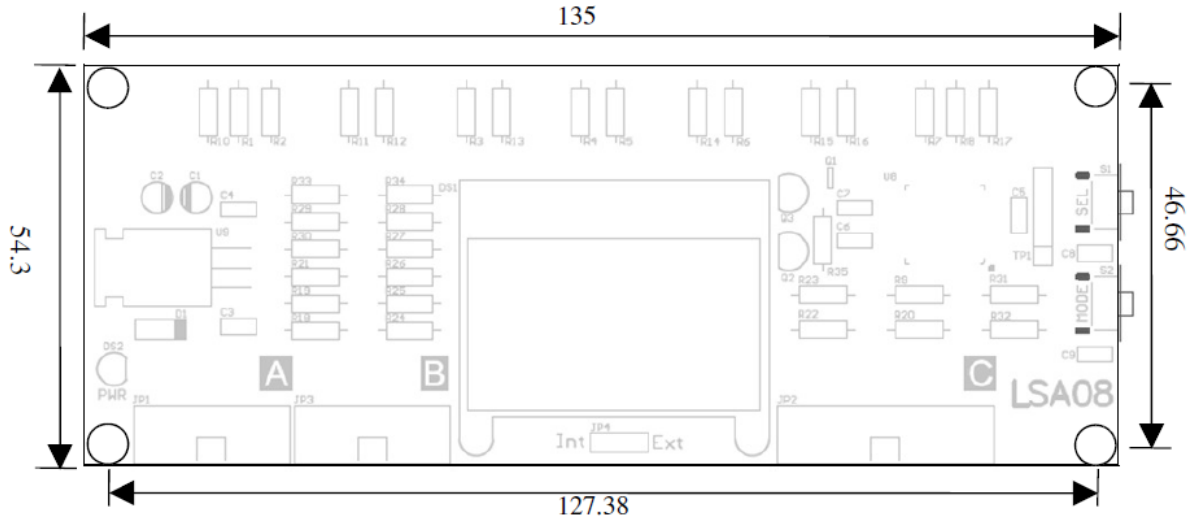
Output Description
Digital - Port 8 bits for 8 sensors
UART- Retrieve digital value, line position or sensor raw value
Analog -Position value in the form of analog voltage.

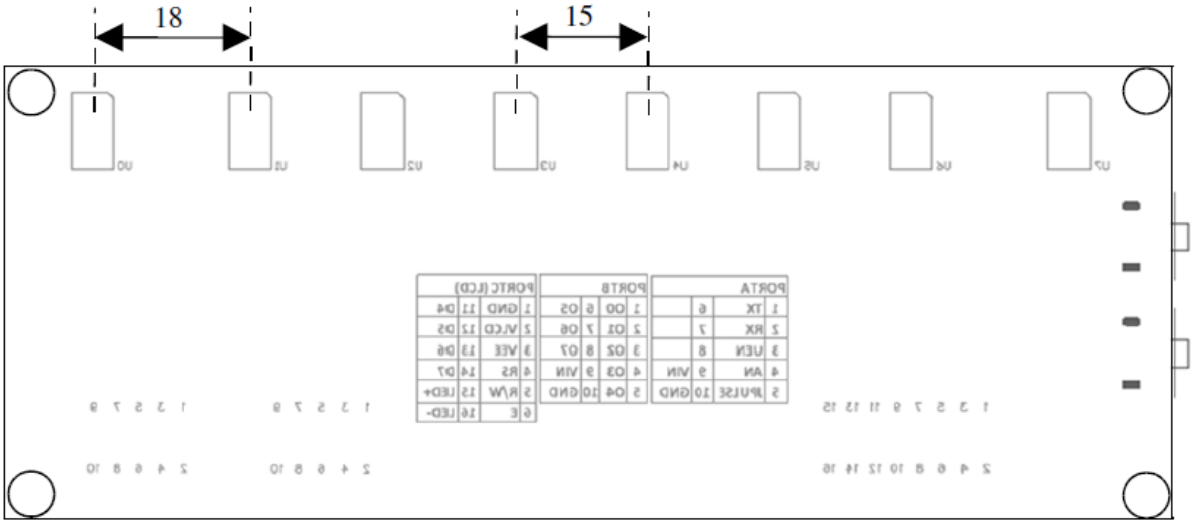
LCD on the LSA08 displays the different menu for setting up the operation setting.
LSA08 have 8 digital outputs to user indicating the existence of the line. The dark and bright value of the line and background will be saved to non-volatile memory when the user calibrate the sensor to the surface that it will recognize. The line existence threshold is set by the Threshold menu of the sensor. Each sensor of the 8 sensors on LSA08 is independent of each other. The refreshing rate of the sensors is more than 100Hz.
LSA08 has a manual mode and select button. User can choose different mode/settings using MODE button and SEL button to enter the mode/settings.

Note: Red color is very bright to IR sensor, thus LSA08 has difficulties in red – white pair color line follow.

Product dimensions and specifications

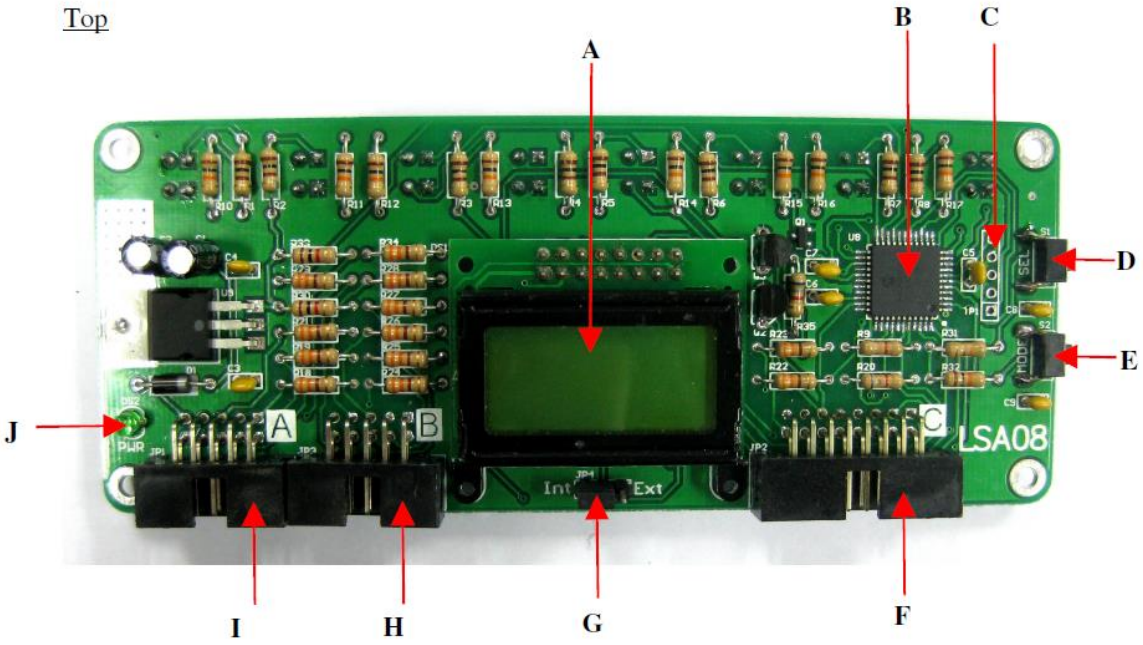






Board layout and parts

Top



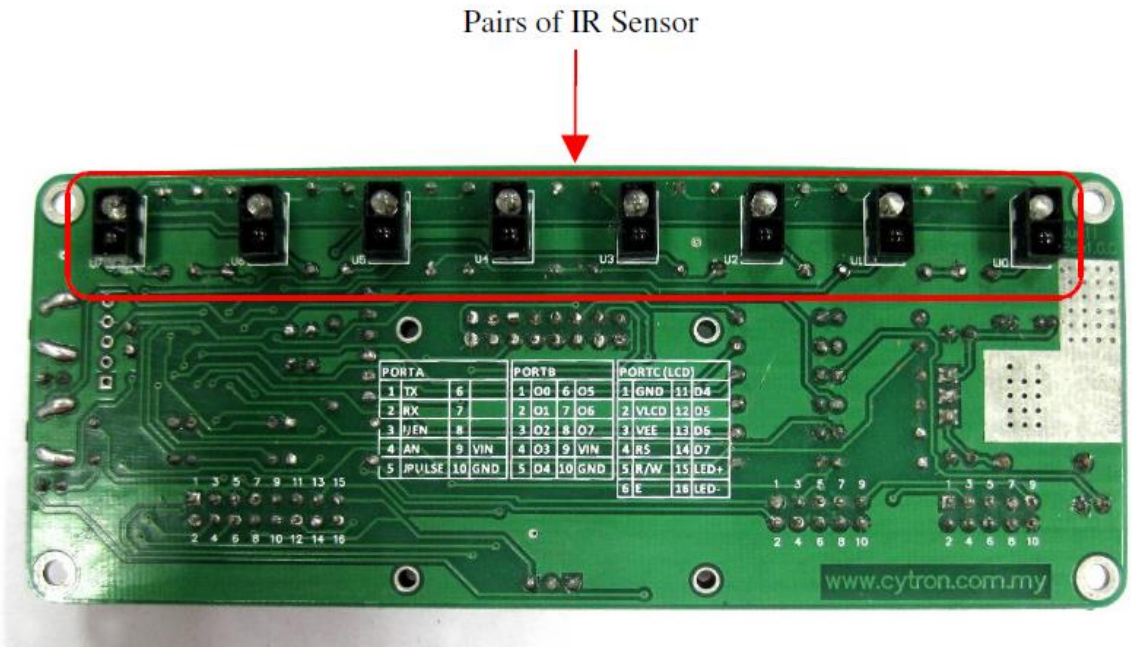
Label	Function	Label	Function
A	2x8 LCD display	F	Port C (External LCD port)
B	PIC16F1937	G	LCD Selector
C	Manufacturing Test Points	H	Port B (Digital port)
D	SEL button	I	Port A (Analog and UART port)
E	MODE button	J	PWR LED

A – 2x8 LCD display to display the different menu for setting up the operation setting.
B – PIC16F1937 PIC microcontroller for data processing.
C – It is reserved for Manufacturing Test Point. Please **DO NOT** short or connect wire to any of these pins.
D – SEL button is use to select modes or setting.
E – MODE button is use to enter different modes/setting.
F – Port C is provided for external LCD port.
G – LCD selector to select either internal or external LCD used. A jumper needs to be set to Ext for the External LCD to operate.

H – Port B provided for digital output port.

I – Port A provided for analog and UART port.
J – Power indicator LED (green) showing the board is supplied with power. Maximum input power is 5V.

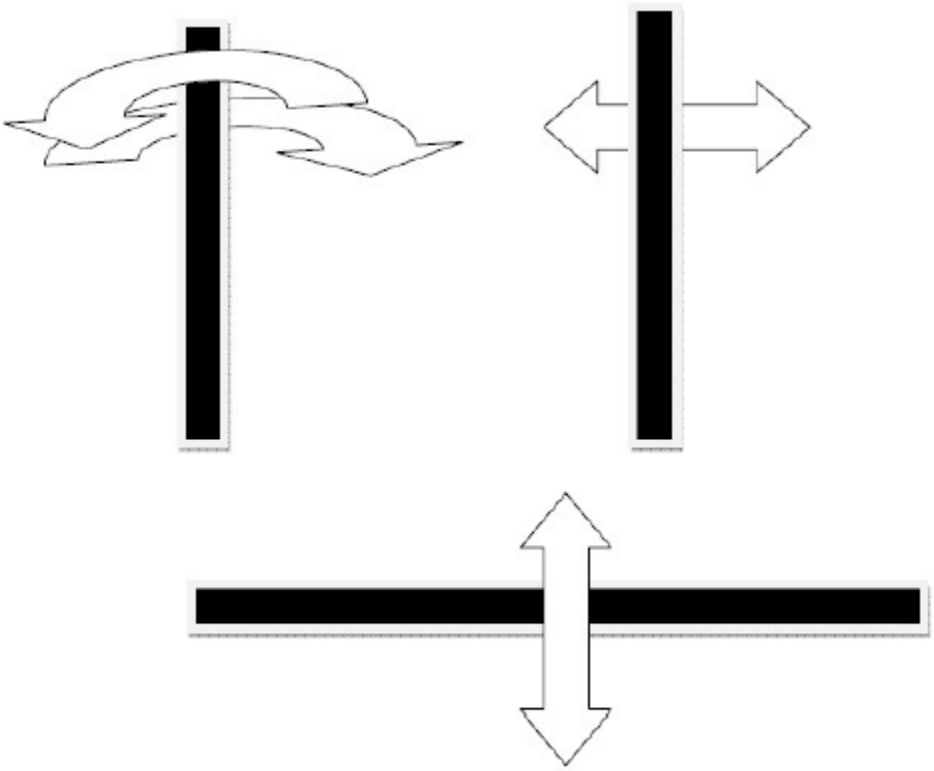
Bottom



GETTING STARTED & Callibration Techniques

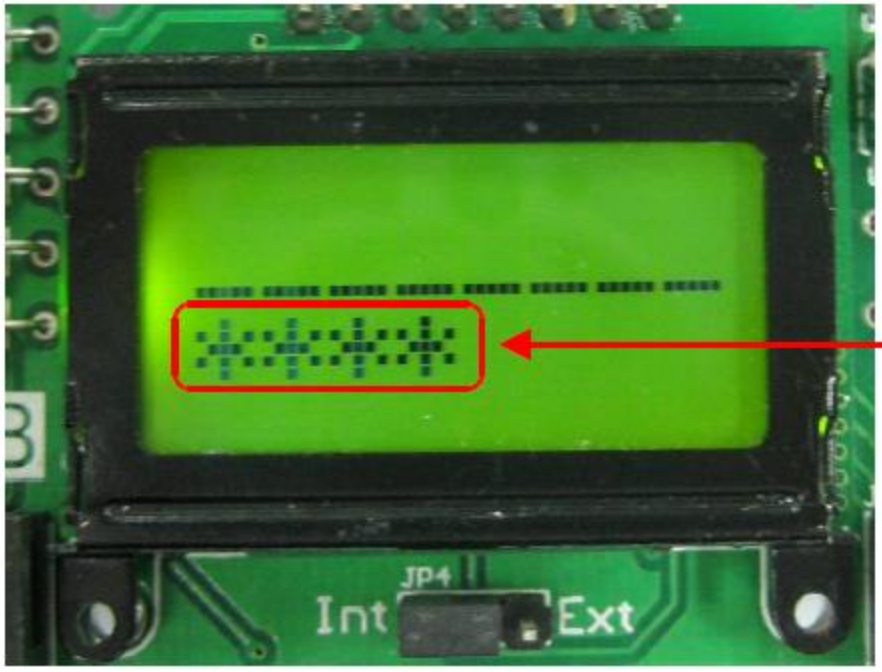
LSA08 need to be calibrated to retrieve the dark and bright value of the surface that it will do the line follow. **Every of the IR sensor pairs** need to be exposed to the dark and bright surface for it to read the value and save it. LSA08 will save the value in EEPROM, it will retrieve back the data from the EEPROM every time its switch on. Hence, only one time calibration is needed for the same background and line unless the background and the line changed, then recalibration is needed. To calibrate LSA08, go into menu setting using MODE button. Choose CALB and enter the mode using SEL button. Calibration is started by

exposing the sensor to the bright surface and then to the dark surface of the line and background in the allocated time. Restart the calibration if missed the timing. LSA08 will save the brightest and darkest value in from the calibration process. User can calibrate by simply swinging the sensors across the dark and bright surface of the line in order to expose every sensor to the dark and bright surface. Calibration of every sensor is independent and value of each sensor will be saved.



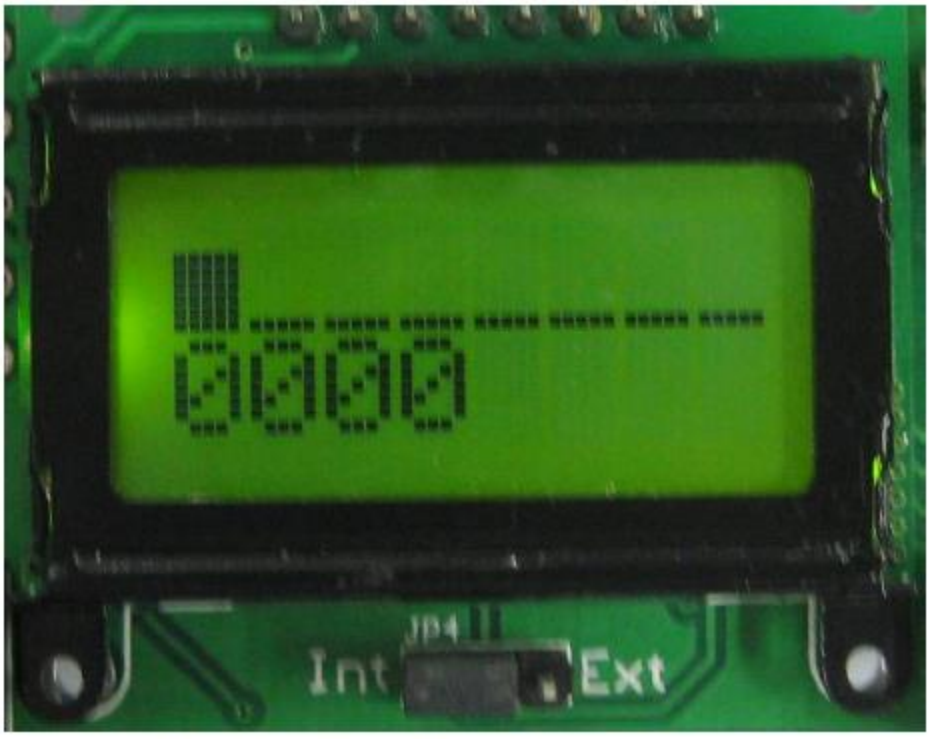
Example motions of calibration by crossing the sensor between the lines.

Once power is supply to LSA08, LCD will display the line position detection and bar chart. Line position detection is a value to shown which sensor is detect the line. The position on LCD display is ranging from 0 to 70. When there is no line detected on LCD, the position value will be shown as "****". If sensor U0 detected a line, the line position reading shown on LCD will be "0000". Likewise if a line is detected on sensor U7 the position shown will be "0070". The position value varies in between linearly.



Line position with no line detected

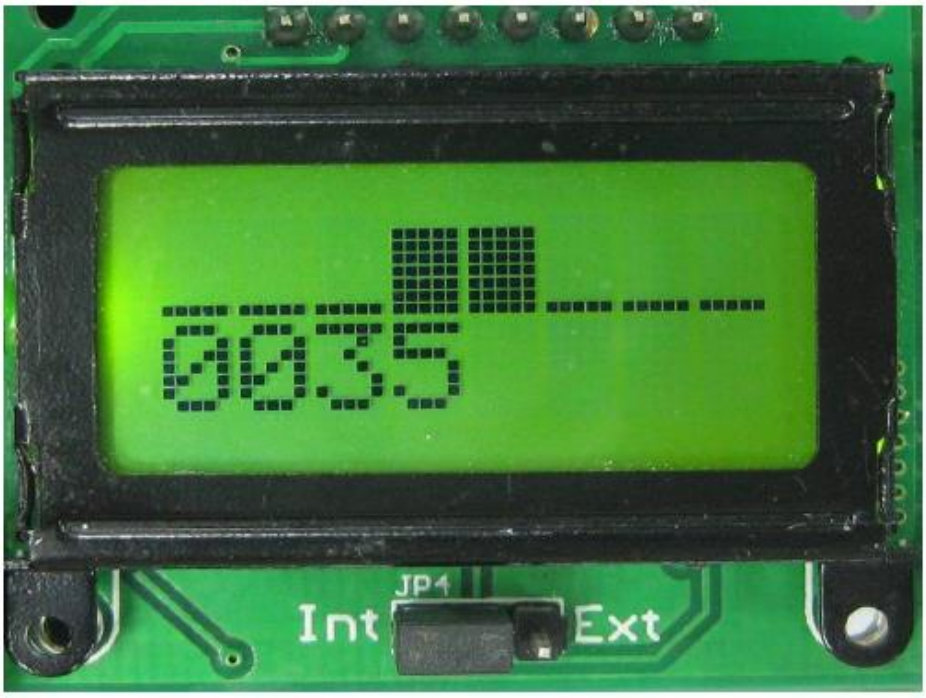
No line detected



Position 00 - U0 detect the dark line

Position 70 - U7 detect the dark line

If a line is at the middle of the LSA08(U3 and U4), the LCD display will show the position as 35.



Position 35 – U3 and U4 detect the dark line

Threshold is the value of bar chart on LCD for LSA08 to treat as a valid line. The threshold value is from 0 to 7. If threshold is set to 7 for example, LSA08 will not detect the line if the bar chart on LCD is less than 7 LCD also will only display “****” means no line detect.
For this example, the threshold is set to 2 and U1 is detect the dark line because the bar line is more than 2.

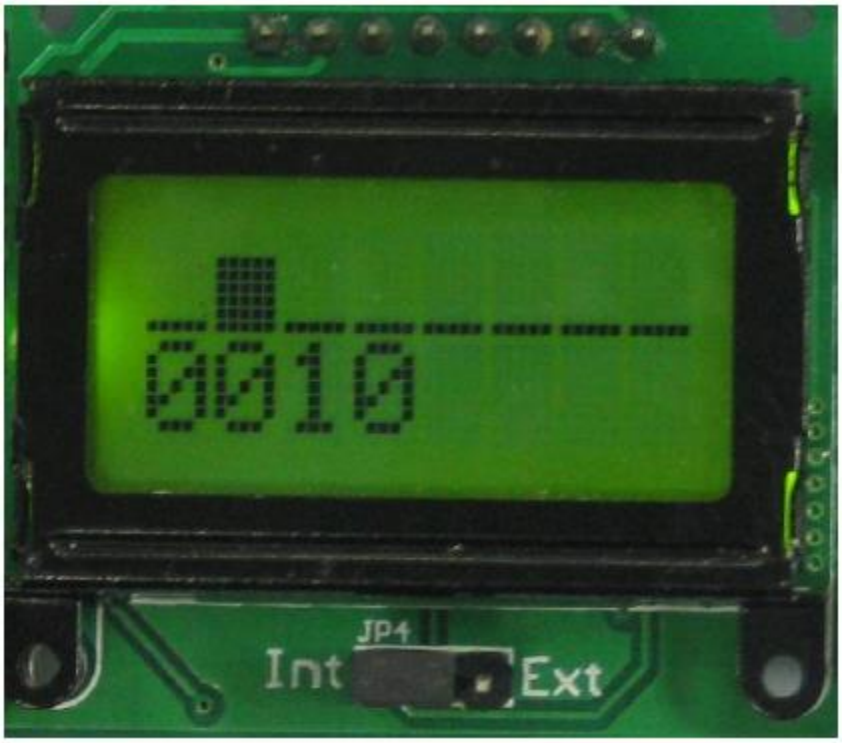
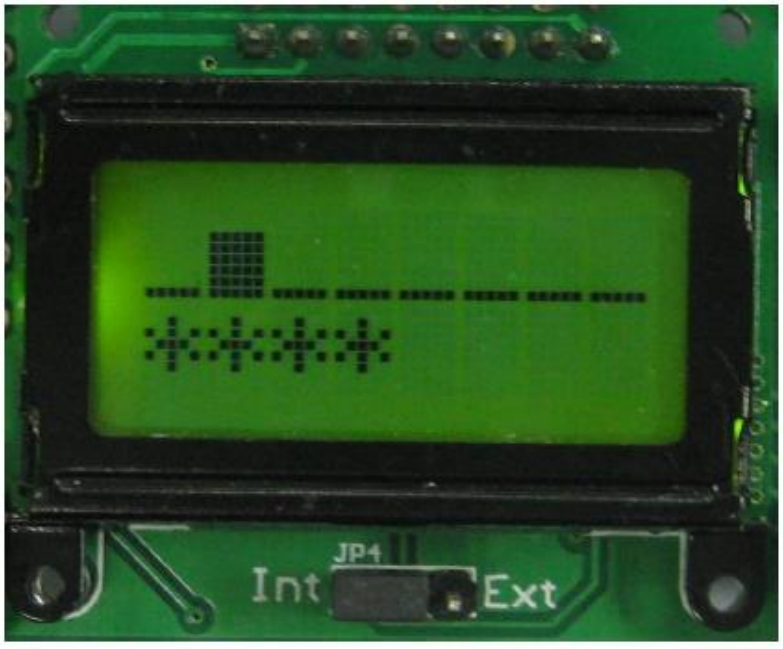


Figure below shown LSA08 cannot detect the line because the threshold value is 7 and the bar line display on LCD is less than 7.



USING CYTRON SENSOR TO MAKE LINE FOLLOWING BOTS

Hardware:

Arduino Uno board

USB B Type Cable

Advanced Auto-Calibrating Line Sensor (LSA08)

Dual Channel Motor Driver (MDD10A)

LiPo battery 11.1V

DC Jack (male)

Ball Castor

DC Geared Motor x 2

Plastic Wheel x 2

Motor Bracket x 2

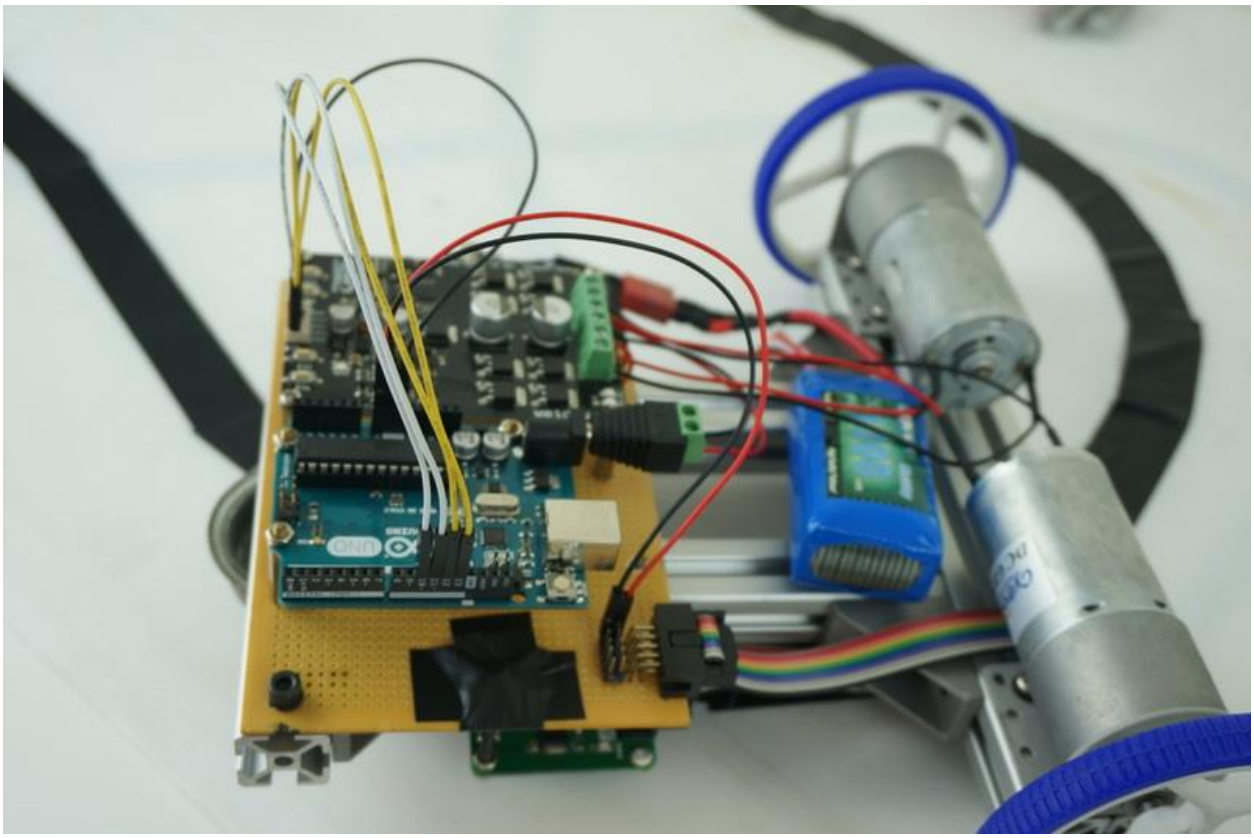
Procedures

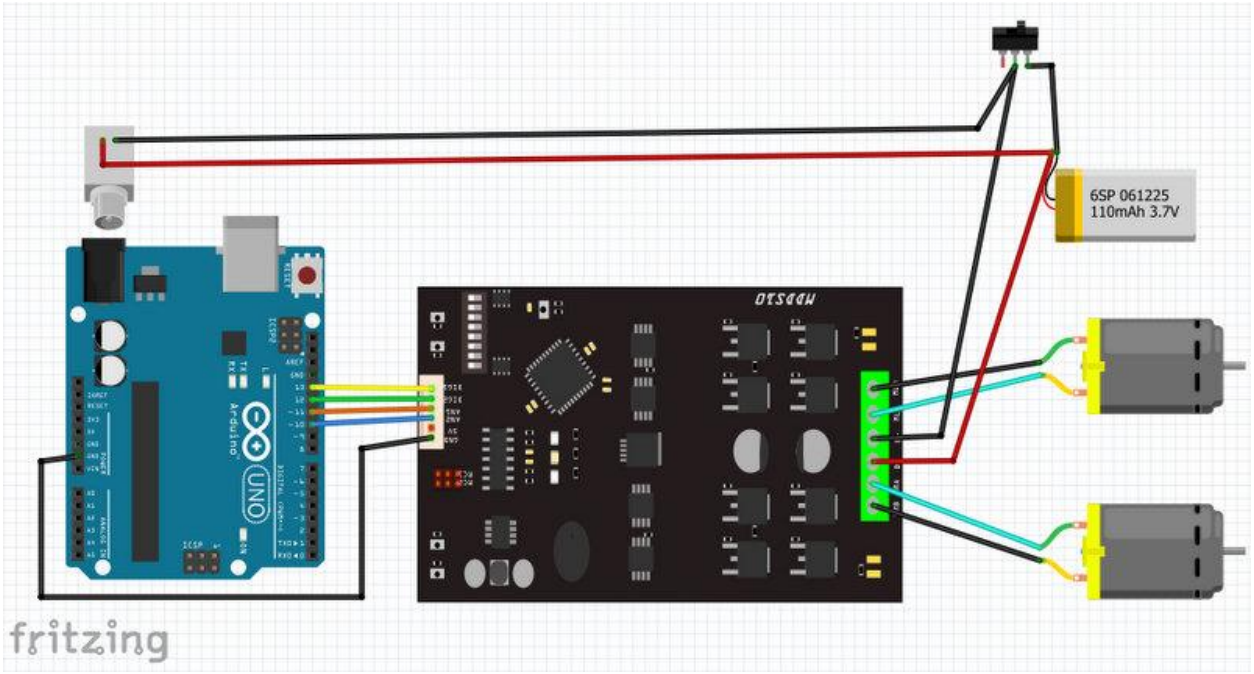
Alright, now we have all we need, let's start by making a robot base. 1…2…3……

Here we go, a robot base~

In case you wondering the steps of robot making, you can view it here. A small precaution here is to make sure that the base is larger than 20cm x 20cm, so you can mount everything we need on it.

So the next thing is to mount everything on it, like this…





Here is the connection that you will need, we will discuss about the connection of LSA08 after this.

Notice that I did not connect 5V pin on MDD10A because there is no such pin on my real board, you have to connect it to 5V pin on your Arduino board if such pin exist.

A switch is highly recommended to avoid your robot running away once you connect the battery =P.

In the end, your connection should looks like the one at left.

Working with LSA08

Figures at the left shows the Advanced Auto-Calibrating Line Sensor (LSA08) line sensor that I always mentioned about. What makes it different with the normal Auto-Calibrating Line Sensor (LSS05) is that it has the word "advanced" in its name (of course).

LSA08 has 8 sensor array rather than 5 in LSS05, besides it also allows the user to manually select the line sensing option through its setting menu.

Menu	Description
LCD Contrast (LCD CON)	Setting the contrast of LCD
Calibration (CALB)	Calibrate LSA08 to the colour brightness of the line and background
Line Mode (LINEMODE)	Setting LSA08 to be Dark On (for dark line and bright background) or Light On (for bright line and dark background)
Threshold (THRES)	
Junction Width (J WIDTH)	Junction width, the number of bar chart on LCD for LSA08 to treat it as a junction crossing.
UART Address (UART ADD)	Unique UART address for the LSA08.
UART Baudrate (BAUDRATE)	Setting the baudrate of the UART Communication
UART Mode (UARTMODE)	Setting the UART output mode
LCD Backlight (LCD B/L)	Setting the LCD backlight brightness
Exit (EXIT)	Exit menu
Exit Menu shortcut	Press and hold Mode button on any main menu will exit the menu

Here is the list of available setting for LSA08. More explanation about the setting menu will be discussed later.

Arduino Code

```
const
byte rx
= 0;
//
Defining
pin 0 as
Rx
```

```
const byte tx = 1;    // Defining pin 1 as Tx
const byte serialEn = 2;    // Connect UART output enable of LSA08 to pin 2
const byte junctionPulse = 4;   // Connect JPULSE of LSA08 to pin 4
const byte dir1 = 13;   // Connect DIR1 of motor driver to pin 13
const byte dir2 = 12;   // Connect DIR2 of motor driver to pin 12
const byte pwm1 = 11;   // Connect PWM1 of motor driver to pin 11
const byte pwm2 = 10;   // Connect PWM2 of motor driver to pin 10


unsigned int junctionCount = 0;   // Variable to store junction count value
```

```
void setup() {
  pinMode(serialEn,OUTPUT);   // Setting serialEn as digital output pin
  pinMode(junctionPulse,INPUT);   // Setting junctionPulse as digital input pin


  // Setting pin 10 - 13 as digital output pin
  for(byte i=10;i<=13;i++) {
    pinMode(i,OUTPUT);
  }


  // Setting initial condition of serialEn pin to HIGH
  digitalWrite(serialEn,HIGH);


  // Setting the initial condition of motors
  // make sure both PWM pins are LOW
  digitalWrite(pwm1,LOW);
  digitalWrite(pwm2,LOW);


  // State of DIR pins are depending on your physical connection
  // if your robot behaves strangely, try changing thses two values
  digitalWrite(dir1,LOW);
  digitalWrite(dir2,LOW);


  // Begin serial communication with baudrate 9600
  Serial.begin(9600);


  // Clear internal junction counter of LSA08
  clearJunction();


}


void loop() {
  byte dummy = 0;   // Declare a dummy variable to store incoming data

  // Checking for junction crossing, if juction detected,
  // keep moving forward
```

```
  if(digitalRead(junctionPulse)) {
    while(digitalRead(junctionPulse)) {
      moveForward();
    }
    // Retrieve the junction count from LSA08
    // You can do whatever you want with the junction count
    junctionCount = getJunction();
  }

  digitalWrite(serialEn,LOW);   // Set serialEN to LOW to request UART data
  while(Serial.available() <= 0);   // Wait for data to be available
  dummy = Serial.read();    // Read incoming data and store in dummy
  digitalWrite(serialEn,HIGH);    // Stop requesting for UART data


  // Checking for sensor number 1 and 2, if line detected, move left
  if(bitRead(dummy,1) || bitRead(dummy,2))
  moveLeft();


  // Checking for sensor number 5 and 6, if line detected, move right
  else if(bitRead(dummy,5) || bitRead(dummy,6))
  moveRight();


  // Checking for sensors number 3 and 4,
  // if line is detected by either of these sensor, move forward
  else if(bitRead(dummy,3)||bitRead(dummy,4))
  moveForward();


  // If no line is detected, stay at the position
  else
  wait();


  // Put some delay to avoid the robot jig while making a turn
  delay(100);


}
```

```
// Function to clear internal junction counter of LSA08
void clearJunction() {
  char address = 0x01;
  char command = 'X';
  char data = 0x00;
  char checksum = address + command + data;


  Serial.write(address);
  Serial.write(command);
  Serial.write(data);
  Serial.write(checksum);
}


// Function to retrieve junction count from LSA08
int getJunction() {
  char address = 0x01;
  char command = 'X';
  char data = 0x01;
  char checksum = address + command + data;


  Serial.write(address);
  Serial.write(command);
  Serial.write(data);
  Serial.write(checksum);


  while(Serial.available() <= 0);
  return (int(Serial.read()));
}


// The values work good in my case, you could use other values set
// to archieve a performance that satisfy you
void moveLeft() {
  // For robot to move left, right motor has to be faster than left motor
  analogWrite(pwm1,90);
  analogWrite(pwm2,10);
}
```

```
void moveRight() {
  // For robot to move right, left motor has to be faster than right motor
  analogWrite(pwm1,10);
  analogWrite(pwm2,90);
}


void moveForward() {
  // For robot to move forward, both motors have to be same speed
  analogWrite(pwm1,70);
  analogWrite(pwm2,70);
}


void wait() {
  // Function to makes the robot stay
  analogWrite(pwm1,0);
  analogWrite(pwm2,0);
}
```

Command test code in serial mode for the Arduino

```
char
address
= 0x01;
// UART
address
as 0x01


void setup() {
  Serial.begin(9600);   // Start serial communication
  Serial.flush();   // Clear serial buffer


}


void loop() {


  char command, data;
```

```
// Clear internal junction count of LSA08
command = 'X';
data = 0x00;
sendCommand(command,data);


// Setting LCD contrast to 90
command = 'S';
data = 90;
sendCommand(command,data);


// Setting LCD backlight to level 5
command = 'B';
data = 0x05;
sendCommand(command,data);


// Setting junction width to 6
command = 'J';
data = 0x06;
sendCommand(command,data);


// Setting threshold value to 5
command = 'T';
data = 0x05;
sendCommand(command,data);


// Setting line mode to Dark-On
command = 'L';
data = 0x01;
sendCommand(command,data);


// Setting UART ouput to mode 1
command = 'D';
data = 0x01;
sendCommand(command,data);
```

```
  // Start calibration
  command = 'C';
  data = 0x00;
  sendCommand(command,data);


  while(1);   // Stay here to prevent infinite loop


}


/*
 * Function to send command to LSA08 in 4 continuous bytes.
 * LSA08 will reply with "OK" for every successful command sent.
 * However, reading the reply is optional, and thus not showing here.
 */
void sendCommand(char command, char data) {

  char checksum = address + command + data;

  Serial.write(address);
  Serial.write(command);
  Serial.write(data);
  Serial.write(checksum);


}
```

The same code can be configured with PID and uploaded to the Arduino to implement the auto learning:

Interfacing with PID controller

Now, let's talk about PID controller where P stands for Proportional, I stands for Integral, and D stands for Derivative. Well, it is nothing but a piece of code that allows us to find the deviation of robot to the line, and correcting the position by changing the motors speed.

The concept is simple, we decide a set point on the sensor array (mostly the middle) and the robot will always try to adjust itself to centered at the set point. LSA08 support range

from 0 – 70, so 35 will be our set point value, and the target of our PID controller is to make sure we achieve the position 35 in shortest time.

So, let's see what we need to do.

First of all, we need to find the error of deviation using the formula

error = current position – set point value

this will yields us negative value when our robot is deviating to the right, or positive value when our robot is deviating to the left.

Next, applying the PID formula to find the required change in speed,

*speed change = Kp * error + Kd * (error – previous error)*

where Kp and Kd are the constants that we need to determine through endless of experiments, while the previous error is the error before this iteration.

Notice that I said PID controller, but in fact we are just applying PD controller, because adding Integral control will not affect the result much.

And here, the last step is to adjust your motors speed by how much it needs to change.

right motor speed = base speed – speed change
left motor speed = base speed + speed change

where, you can decide the base speed of your motors so that you can have your motors move faster or slower. You might need to interchange the + and – sign if your robot move in opposite direction.

```
char
address
= 0x01;
// UART
address
as 0x01
```

```
void setup() {
  Serial.begin(9600);   // Start serial communication
  Serial.flush();   // Clear serial buffer

}

void loop() {

  char command, data;

  // Clear internal junction count of LSA08
  command = 'X';
  data = 0x00;
  sendCommand(command,data);

  // Setting LCD contrast to 90
  command = 'S';
  data = 90;
  sendCommand(command,data);

  // Setting LCD backlight to level 5
  command = 'B';
  data = 0x05;
  sendCommand(command,data);

  // Setting junction width to 6
  command = 'J';
  data = 0x06;
```

```
  sendCommand(command,data);


  // Setting threshold value to 5
  command = 'T';
  data = 0x05;
  sendCommand(command,data);


  // Setting line mode to Dark-On
  command = 'L';
  data = 0x01;
  sendCommand(command,data);


  // Setting UART ouput to mode 1
  command = 'D';
  data = 0x01;
  sendCommand(command,data);


  // Start calibration
  command = 'C';
  data = 0x00;
  sendCommand(command,data);


  while(1);   // Stay here to prevent infinite loop


}


/*
 * Function to send command to LSA08 in 4 continuous bytes.
 * LSA08 will reply with "OK" for every successful command sent.
 * However, reading the reply is optional, and thus not showing here.
 */
void sendCommand(char command, char data) {

  char checksum = address + command + data;

  Serial.write(address);
```

```
    Serial.write(command);
    Serial.write(data);
    Serial.write(checksum);


}
```

We can buy the cytron line sensor at:

http://www.amazon.in/Advanced-Auto-Calibrating-Line-Sensor-LSA08/dp/B01MT6ODGG?tag=shop099-21

Troubleshooting:

Finally, upload the sketch to Arduino board, and run the program. If your robot moving in a wrong way, try changing the polarities of your motor.

There are several ways on how to get the Kp and Kd value, and I found this is the most understandable:

First increasing the Kp value while maintain Kd as 0, until a condition where your robot move fast enough and without causing it to slip.

Then, divide the Kp value by half, and set the Kd value to the same as Kp, and from here, increasing the Kd value while maintain Kp value, until a condition where your robot move fast enough and without causing it to slip.

*NOTE: Kd > Kp always. And please don't ask for the values, as there is no exact values for this, values from the others might not suit in your case.

**NOTE: You can analog read potentiometers instead of software tuning the Kp and Kd values, it saves you A LOT of time T^T.