Particle Detector (Personal)

1. Objective

The particle detection module consisted of an SDS011 dust matter detector which calculated concentrations of PM2.5 and PM10. The calculated concentrations were sent to an Arduino Uno where the signal was processed, and values displayed on an LCD. The architecture of this unit is shown within Figure 1. The main purpose of this unit was to detect the presence of dust or water droplets.

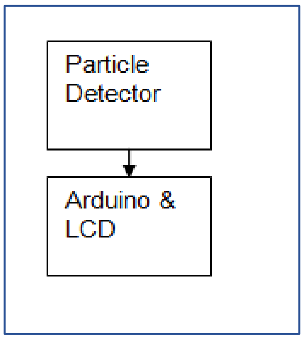


Figure 1: Minimalistic detector architecture

2. Components

2.1 SDS011 Sensor

The SDS011 sensor from figure 2 worked on the principle of laser scattering with a digital output and measured concentrations of particles with mean diameters from 0.3 to 10 µm. Its range was 0-999 µg/m^3 and it had a resolution of 0.3 µg/m^3. It utilised a UART method of communication and was easy to integrate with an Arduino. The relative error of the sensor was a maximum of ±15% or ±10ug/m3. The serial data output frequency was fixed at 1Hz.

Figure 2: SDS011 sensor

2.2 LCD-016M002B

This device was chosen to display concentration values of the particles detected. This display consisted of 2 lines with 16 characters within each. Each character was made of a 5x8 dot matrix. The DDRAM address for each display position ranged from 00 to 4F consecutively. The built-in driver and controller "KS0066" enabled compatibility with Arduino microcontrollers and the native library "LiquidCrystal.h". The pin configuration table may be found within appendix C and a wiring diagram in subsequent sections. The rated voltage for the backlight and supply was 5V.

2.3 Arduino Uno Rev 3

The Uno Rev 3 board was used as a control unit for the detection module. The microcontroller available was an ATMega328P with a 16MHz clock cycle. The method used to supply power to the board was a USB cable from a PC. This also served to flash on programs using the pre-existing bootloader and USB to serial conversion. The main functionalities utilised were timer interrupts, software serial libraries, and digital IO pins.

Figure 3: Arduino Uno

3. Wiring Schematic

The wiring of this module followed the specification sheets provided and the schematic is as shown in figure 4.

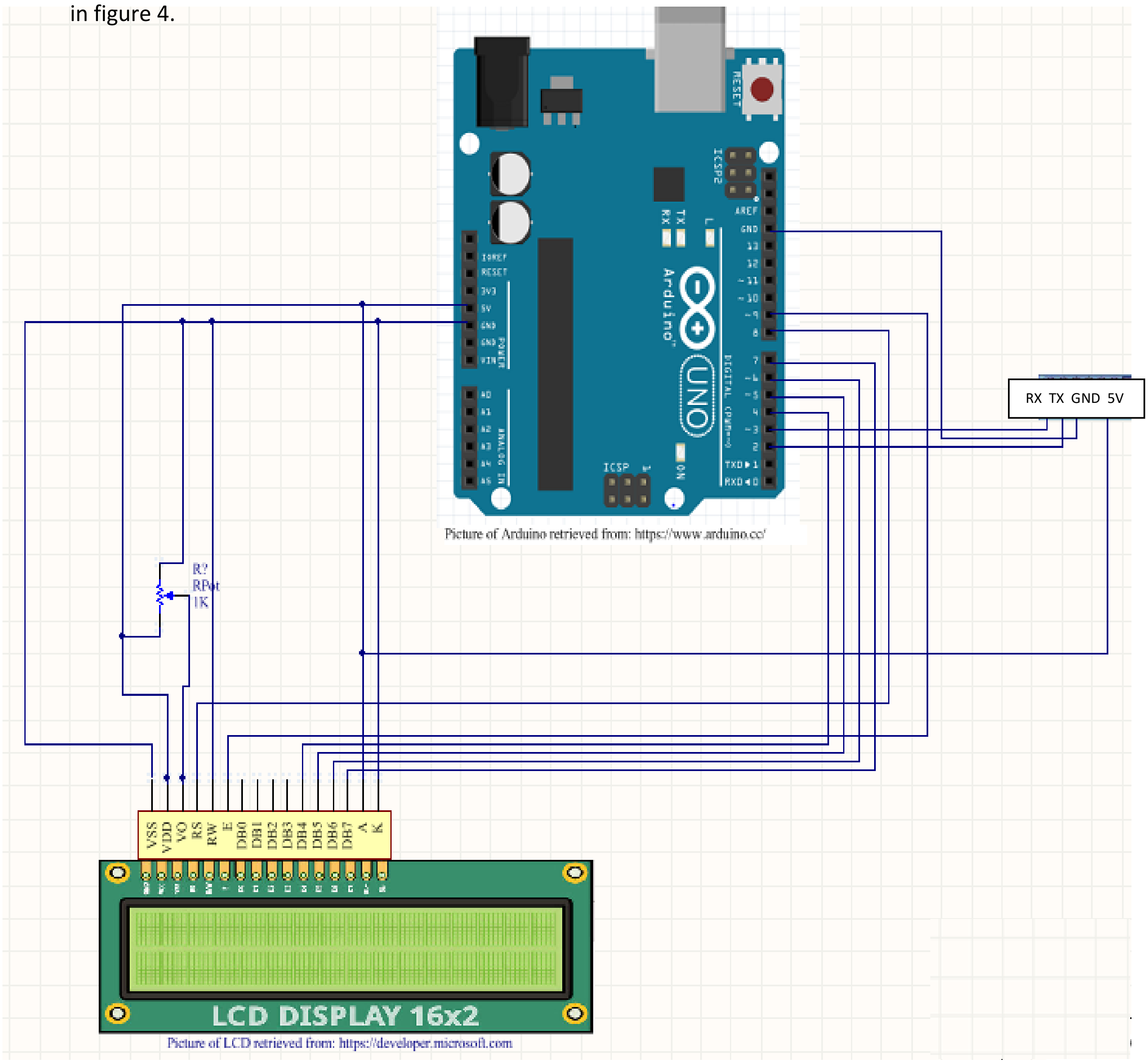

Arduino
UNO
Picture of Arduino retrieved from: https://www.arduino.cc/
R?
RPot
1K
RX TX GND 5V
VSS
VDD
VO
RS
RW
E
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
A
K
LCD DISPLAY 16x2
Picture of LCD retrieved from: https://developer.microsoft.com

Figure 4: Wiring schematic of LCD, SDS011, and Arduino made on Altium

Both the LCD and SDS011 sensor required a connection to the 5V pin for power supply with the LCD having three separate terminals requiring a connection to it. A miniature proto board was utilised for the 5V pin and the separate ground connections together with the placement of the trim potentiometer for contrast adjustment. Digital IO pins 4-9 were used for the LCD with 8 and 9 being the write and enable pins respectively. Pins 2 and 3 were used for RX and TX corresponding to the TX and RX terminals on the SDS011. This was utilised together with the software serial library. The reason for this was pin 1, the original RX pin, was utilised by the bootloader for flashing on programs.

4. Software Design

Arduino IDE which is based on AVR C++/C was the environment in which all programs were coded. The two native libraries which were extensively utilised were: "LiquidCrystal.h" and "SoftwareSerial.h".

4.1 LCD Functionality

The previously mentioned library was object oriented and so an instance was made with pins 4, 5, 6, 7, 8 and 9. The display for both is as shown in figure 5 with PM2.5 values on the first line and PM10 values on the second.

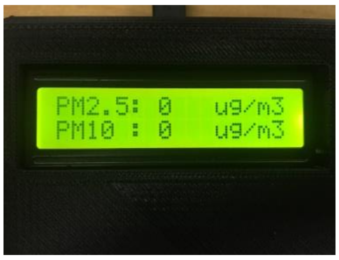


Figure 5: Displaying PM10 and PM2.5 values

4.2 SDS011 Functionality

The "SoftwareSerial.h" library allowed pins 2 and 3 to mimic the functionality of RX and TX pins. The serial read functionality was implemented within an interrupt vector. The overflow interrupt occurred every 0.9 seconds and was made using a prescalar of 256 and a 16-bit timer. A period close to 0.9 seconds was chosen as the hardware refresh rate was a second. The preloading of the timer used the relationship found in equation 1 where "T" is the period and "Preload" is the number of ticks till the interrupt subtracted from the number of ticks in the timer.

$$Preload = 2^{16} - \frac{16MHz}{256}T \tag{1}$$

An array of 10 elements held each packet of 10 bytes with checks done for data stability. There was no parity bit available within the packet. The check-sum byte was constantly evaluated with equation 2.

$$Checksum = \sum_{N=1}^{6} DataN \tag{2}$$

The values for PM2.5 and PM10 concentrations were calculated using their high and low bytes. Information on each byte and their description is found within appendix B. The concentrations were calculated and displayed from equations 3 and 4 with units of µg/m^3

$$PM2.5\ Value = \frac{(PM2.5\ high\ byte)(\ 2^8)+PM2.5\ low\ byte}{10} \tag{3}$$

$$PM10\ Value = \frac{(PM10\ high\ byte)(\ 2^8)+PM10\ low\ byte}{10} \tag{4}$$

4.3 FIR filter (Moving average)

Due to ripples present at steady state, the displayed value would fluctuate leading to unstable measurements. To alleviate this problem, a simple moving average filter was designed and implemented with the graph showing comparisons of filtered and unfiltered data shown in Figure 6. The number of data points for an average was originally 15 which smoothed out sharp transitions, but ripples were still present as shown in the figure. A filter with 30 data points or more is advised but within reason. More data points collected means a higher response settling time. The MATLAB graphing program is within appendix D.

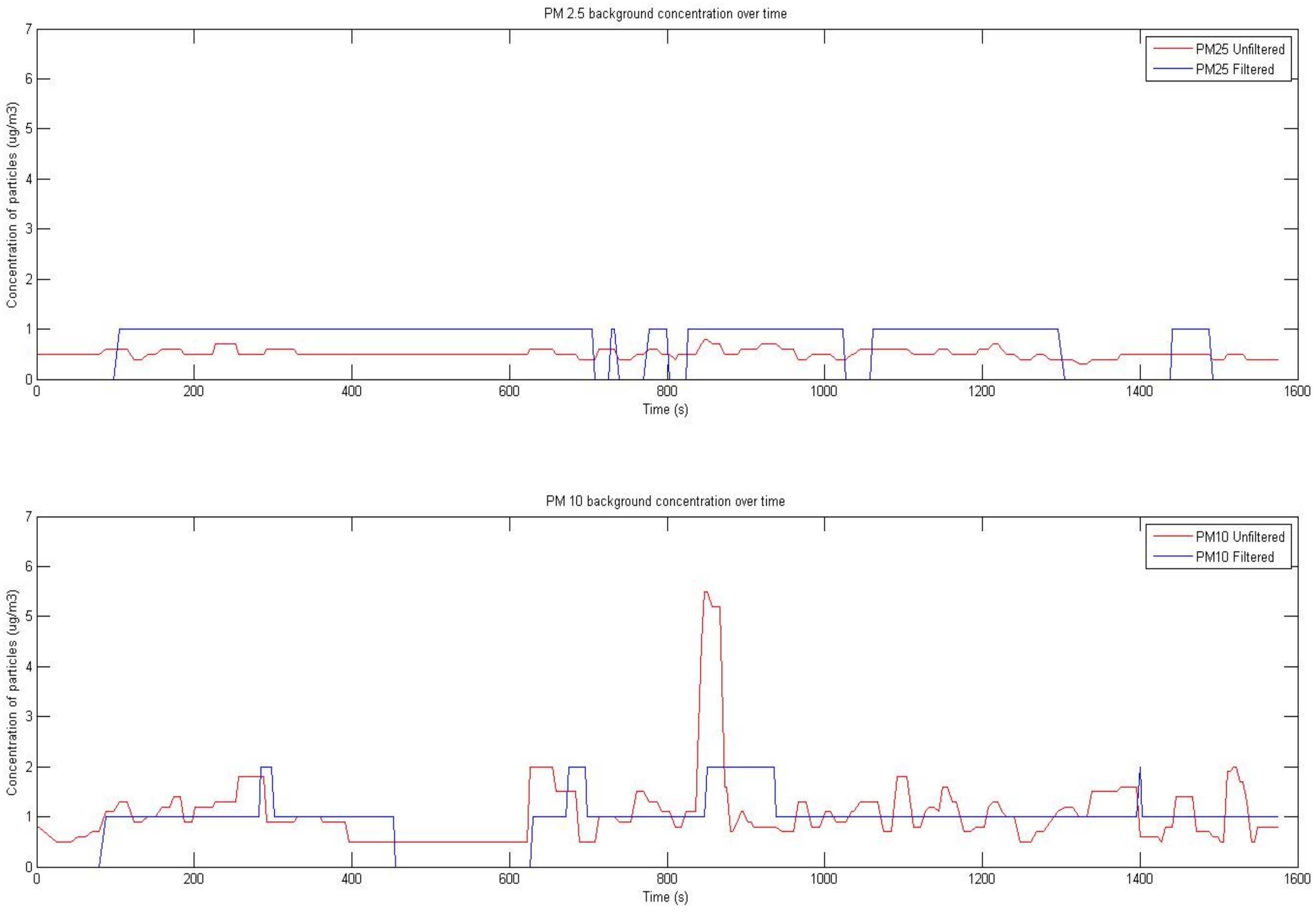


Figure 6: Two graphs of PM2.5 and PM 10 data of both filtered and unfiltered results

5. Enclosure Design

The enclosure was designed around the three main components with special care taken for the dimensions of the SDS011 sensor. The top cover and bottom shell both had numerous dimensions to account. Both the cover and bottom shell were printed with ABS material on an FDM printer. An assembly for both components together with models depicting an LCD and the arduino are shown in Figures 7 and 8.

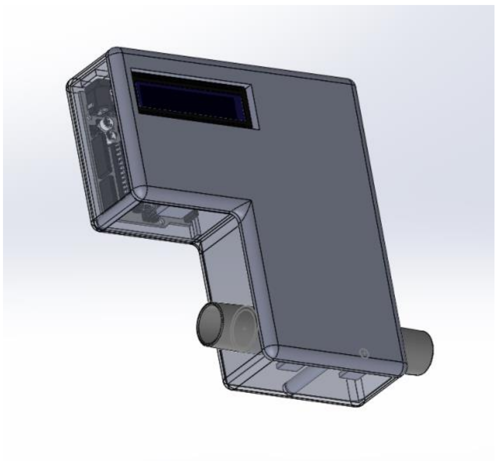


Figure 7: Assembly view to show flow pathway, LCD, and arduino

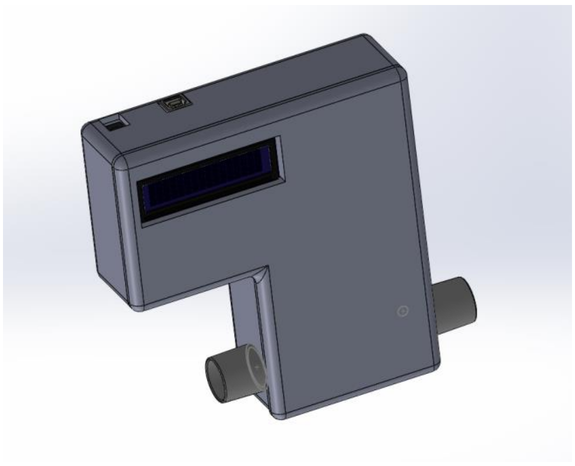


Figure 8: Assembly view showing USB and DC ports

5.1 Top Cover

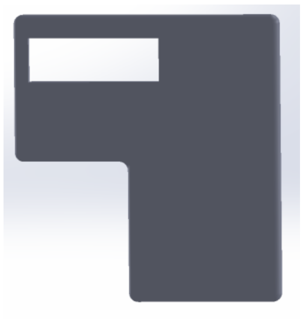


Figure 9: Front top view of cover

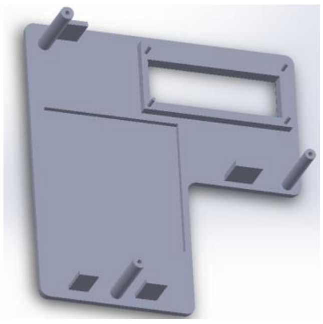


Figure 10: Bottom view of cover showing extrusions.

The top cover has a slot for the LCD measured at 72.5mm x 24.5mm with four extrusions that enable a fit as seen in Figures 9 and 10. Three long extrusions are present for screws to hold the top enclosure to the bottom.

5.2 Bottom shell

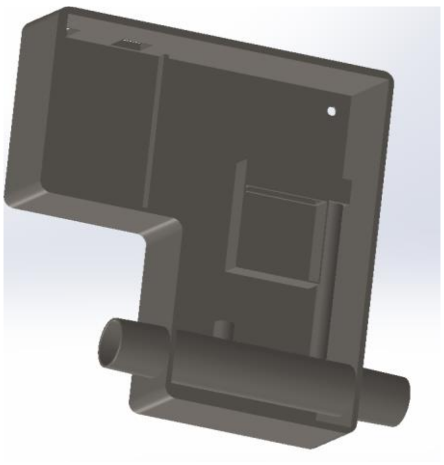


Figure 11: Bottom shell model

The bottom shell consisted of three major sections. The first is the area where the arduino would be placed, the second a slot for the SDS011 sensor and its fan, and third a flow pathway for entrance and exit of airflow containing moisture. The functional diameters for the flow pathway were both 22.5mm tapering off to 21.5mm for fittings. The hose outlet as shown in Figure 11 had an inner diameter of 6mm and an external diameter of 8mm for attachment to the sensor. The opening for the USB port which was primarily utilised was measured to be 11.7mm x 13mm.

The overall dimensions of the envelope for the entire module were measured to be 160mm x 148mm x 48mm.

6. Results

The module enclosure designed was firm and housed the components while providing a flow sample to the sensor. Figure 12 shows the enclosure with the components powered up. The screws held together the casing well. The port for the USB is functional and holds the cable in place.

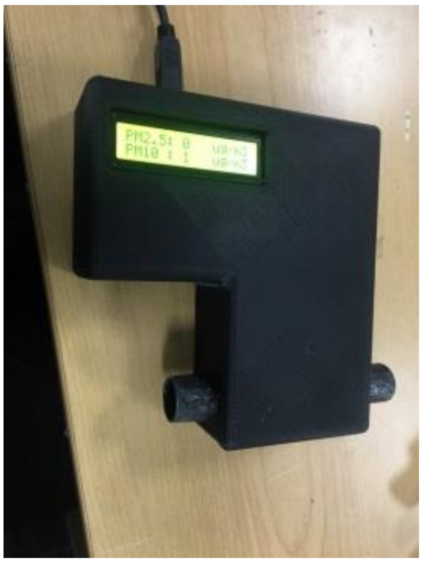


Figure 12: Particle detection module

As seen from figure 13, the graph shows detection of water droplets when atomised water was introduced was 1350 seconds from start. The gradient is lower for the filtered plot compare to the unfiltered plot and response time is clearly slower as was mentioned.

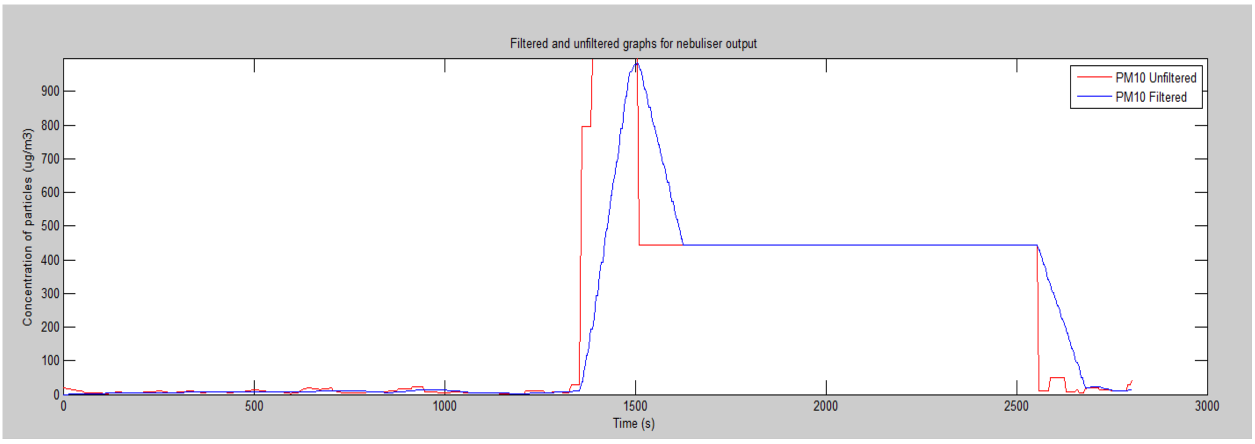
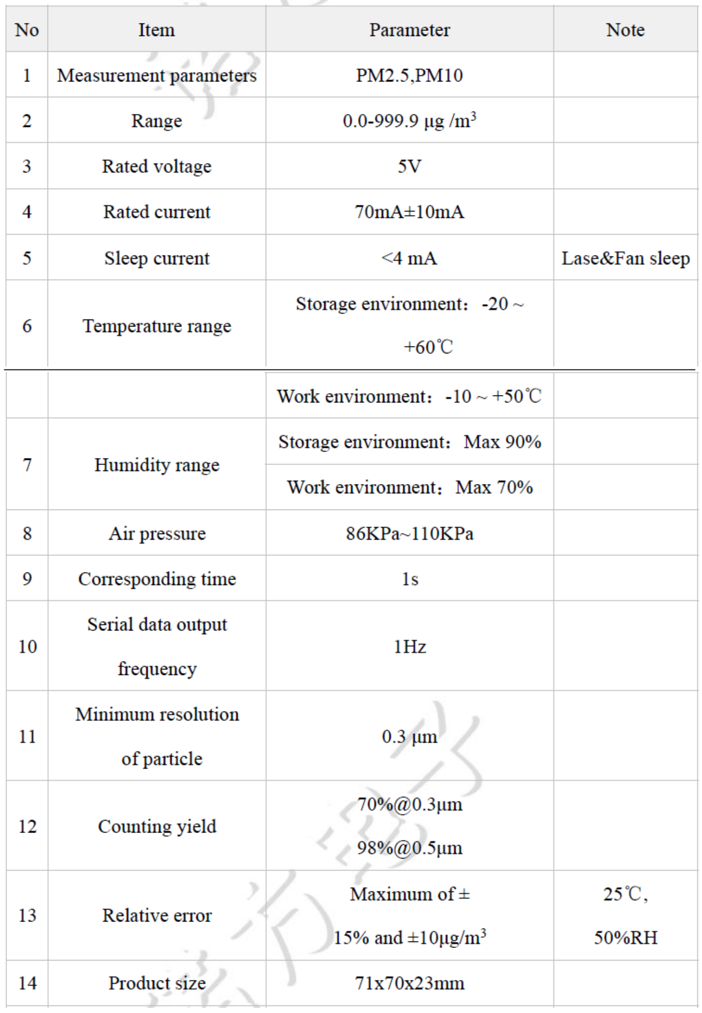


Figure 13: Graph of particle detection module while nebuliser was on

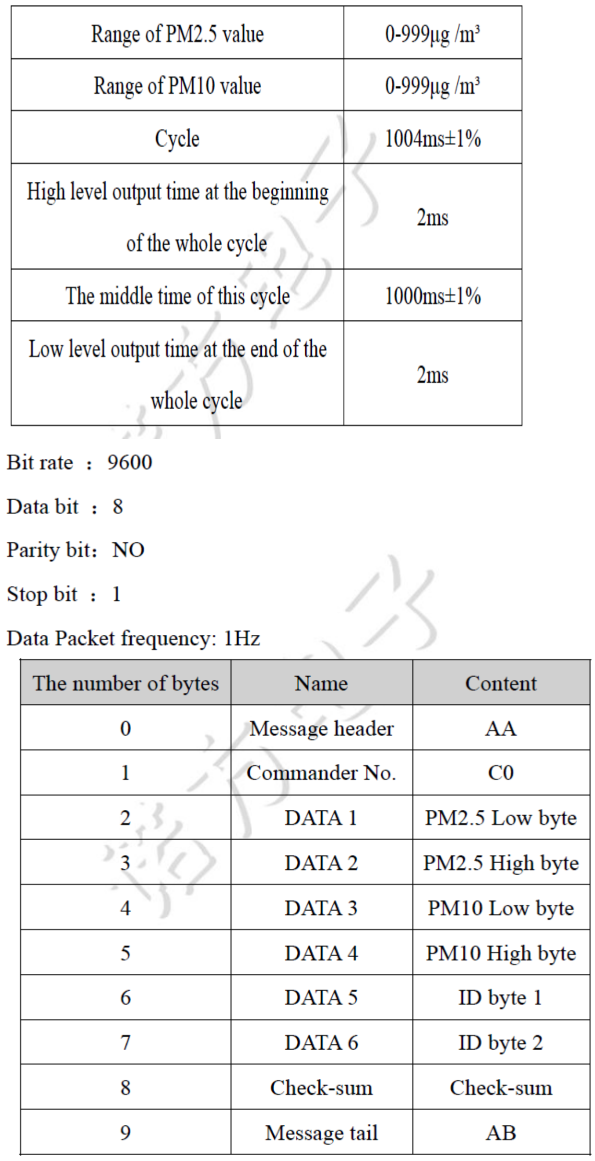
Appendix A: Software for detection module

```
#include <Wire.h>
#include<LiquidCrystal.h>
#include<SoftwareSerial.h>


SoftwareSerial portOne(2,3);
/////////Filter variables///
const int Datapoints = 30;  //CHANGE THIS VARIABLE TO CONTROL FILTER ACCURACY

float PM10Readings[Datapoints];
float PM25Readings[Datapoints];
int Index;
float TotalOutputPm10 =0;
int AverageOutputPm10 =0;
float TotalOutputPm25 =0;
int AverageOutputPm25 =0;
int counter = 0;
int counter2 = 0;
///////////////////////////

////////////PM Sensor variables//////
unsigned volatile int timer1_counter;
float Pm25 = 0;
float Pm10 = 0;
/////////////////////////////////////

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);  //using pins for RS, enable, 4,5,6,7)

////////////Function to process serial data within timer
interrupt////////////
ISR(TIMER1_OVF_vect)        // timer interrupt
{
  uint8_t Data = 0; //holding value of byte
  uint8_t i = 0;
  uint8_t Pkt[10] = {0};  //array for entire packet
  uint8_t Check = 0;      //for checksum

  //from data sheet "AA C0 PM25_Low PM25_High PM10_Low PM10_High 0 0 CRC AB"
    portOne.begin(9600);
    Data = portOne.read();
    delay(2);   //wait  until packet goes through
    if (Data == 0xAA) //head ok
    {
      Pkt[0] =  Data;
      Data = portOne.read();
      if (Data == 0xc0) //commander no. ok
      {
        Pkt[1] =  Data;
        Check = 0;
        for (i = 0; i < 6; i++) //data recieved and crc calculated
         {
          Pkt[i + 2] = portOne.read();
          delay(2);
         Check += Pkt[i + 2];   //Check-sum: Check-sum=DATA1+DATA2+...+DATA6
         }
        Pkt[8] = portOne.read();
        delay(1);
        Pkt[9] = portOne.read();
        if (Check == Pkt[8]) //crc ok
         {
          portOne.flush();
          //Serial.write(mPkt,10);

          Pm25 = (uint16_t)Pkt[2] | (uint16_t)(Pkt[3] << 8);        //PM2.5
value: PM2.5 (μg /m3) = ((PM2.5 High byte *256) + PM2.5 low byte)/10
          Pm10 = (uint16_t)Pkt[4] | (uint16_t)(Pkt[5] << 8);        //PM10
value: PM10 (μg /m3) = ((PM10 high byte*256) + PM10 low byte)/10
          if (Pm25 > 9999)
            Pm25 = 9999;
          if (Pm10 > 9999)
            Pm10 = 9999;

          return;
         }
       }
     }
    TCNT1 = timer1_counter;
    counter++;
    counter2++;
}



/////////////////////////////////////////////////////////////

 void setup() {
  noInterrupts();           // disable all interrupts
  ///////////LCD set up
  lcd.begin(16, 2);
  TCCR1A = 0;
  TCCR1B = 0;

  int Period = 1;
  timer1_counter = (65536-(62500*Period));
  TCNT1 = timer1_counter;   // preload timer
  TCCR1B |= (1 << CS12);    // 256 prescaler
  TIMSK1 |= (1 << TOIE1);   // enable timer overflow interrupt

  for (int i = 0; i < Datapoints; i++) {
    PM10Readings[i] = 0;
    PM25Readings[i] = 0;
  }
    ///////////Sensor set up
  Serial.begin(9600);
  portOne.begin(9600);
  //Serial.begin(9600);
  Pm25 = 0;
  Pm10 = 0;
  interrupts();             // enable all interrupts

}

void loop() {
  lcd.clear();

  if(counter > 3){
  TotalOutputPm10 = TotalOutputPm10 - PM10Readings[Index];
  TotalOutputPm25 = TotalOutputPm25 - PM25Readings[Index];
  // read from the sensor:
  PM10Readings[Index] = (Pm10/10);
  PM25Readings[Index] = (Pm25/10);
  // add the reading to the total:
  TotalOutputPm10 = TotalOutputPm10 + PM10Readings[Index];
  TotalOutputPm25 = TotalOutputPm25 + PM25Readings[Index];
  // advance to the next position in the array:
  Index++;

  // if we're at the end of the array
  if (Index>= Datapoints) {
    //wrap around to the beginning:
    Index = 0;
  }

  // calculate the average:
  AverageOutputPm10 = round(TotalOutputPm10/Datapoints);
  AverageOutputPm25 = round(TotalOutputPm25/Datapoints);
  if(AverageOutputPm10>999){
    AverageOutputPm10 = 999;
  }
  if(AverageOutputPm25>999){
    AverageOutputPm25 = 999;
  }
  delay(1);
  counter = 0;
  }

  lcd.setCursor(0, 0);
  lcd.print("PM2.5: ");
 lcd.print(AverageOutputPm25);
  lcd.setCursor(11, 0);
 lcd.print("ug/m3");
  lcd.setCursor(0, 1);
  lcd.print("PM10 : ");
  //lcd.print(int(Pm10) / 10);
 lcd.print(AverageOutputPm10);
  lcd.setCursor(11, 1);
 lcd.print("ug/m3");


  if(counter2 > 2){
 Serial.begin(9600);
 Serial.print(millis());
  Serial.print(", ");
  //Serial.print("Pm2.5 ");
  Serial.print(float(Pm25) / 10.0);
  Serial.print(", ");
  //Serial.print("Pm10 ");
  Serial.print(float(Pm10) / 10.0);
  Serial.print(", ");
  //Serial.print("Average25 ");
 Serial.print(AverageOutputPm25);
  Serial.print(", ");
  //Serial.print("Average10 ");
 Serial.print(AverageOutputPm10);
  Serial.println();
  counter2 = 0;
  }
  delay(200);
}
```

Appendix B: SDS011 sensor specifications and basic dimensions



No	Item	Parameter	Note
1	Measurement parameters	PM2.5,PM10	
2	Range	0.0-999.9 μg /m^3	
3	Rated voltage	5V	
4	Rated current	70mA±10mA	
5	Sleep current	<4 mA	Lase&Fan sleep
6	Temperature range	Storage environment：-20～+60℃	
		Work environment：-10～+50℃	
7	Humidity range	Storage environment：Max 90%	
		Work environment：Max 70%	
8	Air pressure	86KPa~110KPa	
9	Corresponding time	1s	
10	Serial data output frequency	1Hz	
11	Minimum resolution of particle	0.3 μm	
12	Counting yield	70%@0.3μm 98%@0.5μm	
13	Relative error	Maximum of ± 15% and ±10μg/m^3	25℃, 50%RH
14	Product size	71x70x23mm	



Range of PM2.5 value	0-999μg /m^3
Range of PM10 value	0-999μg /m^3
Cycle	1004ms±1%
High level output time at the beginning of the whole cycle	2ms
The middle time of this cycle	1000ms±1%
Low level output time at the end of the whole cycle	2ms

Bit rate ： 9600

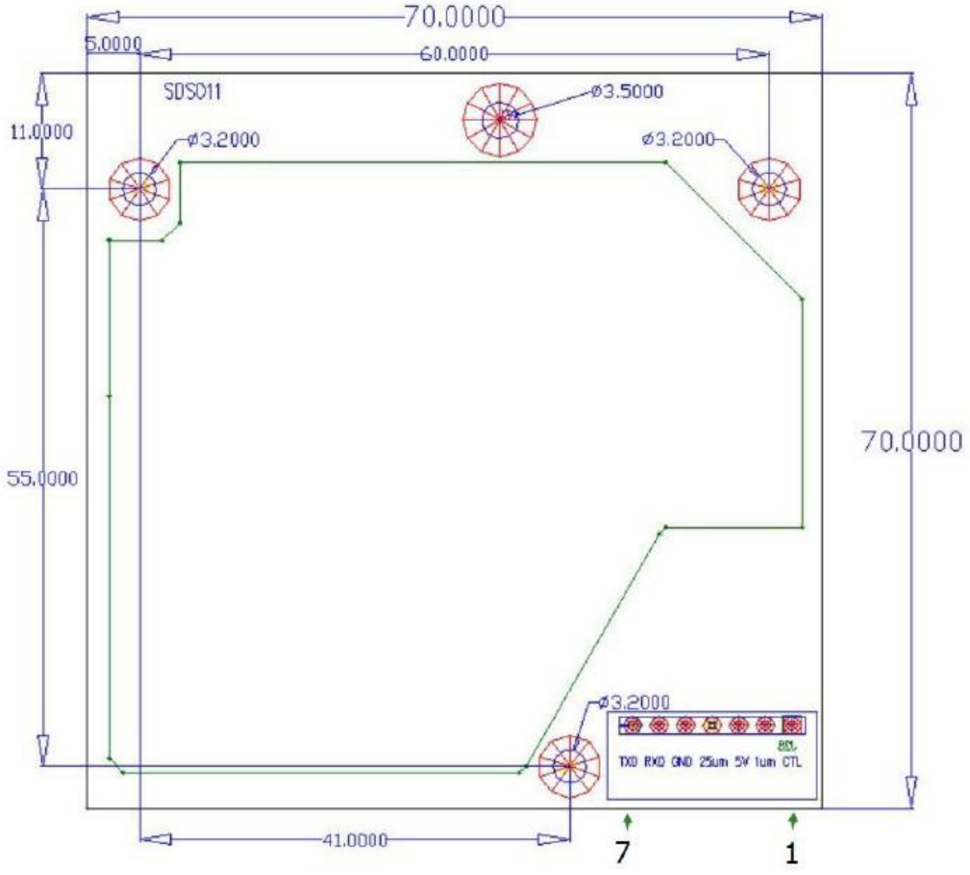
Data bit ： 8

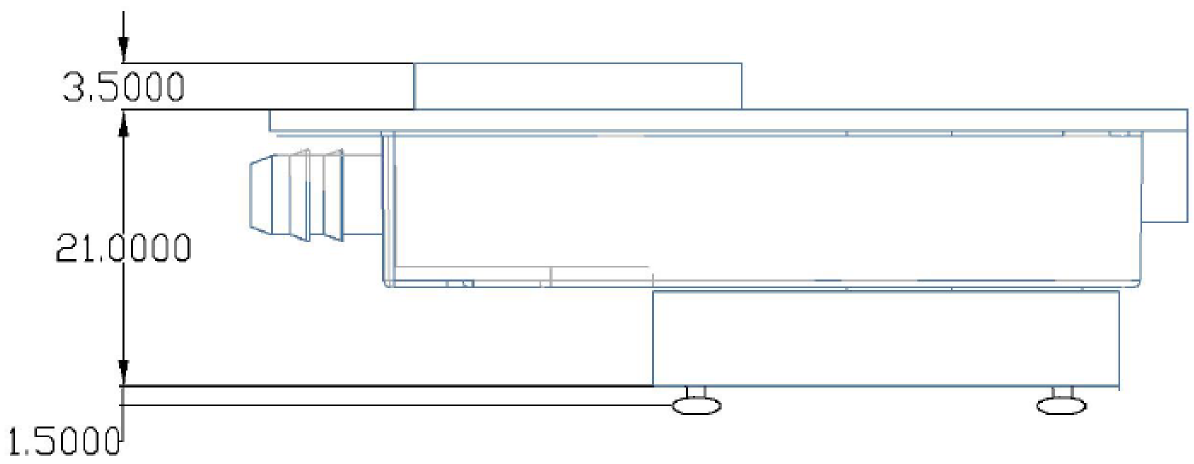
Parity bit：NO

Stop bit ： 1

Data Packet frequency: 1Hz

The number of bytes	Name	Content
0	Message header	AA
1	Commander No.	C0
2	DATA 1	PM2.5 Low byte
3	DATA 2	PM2.5 High byte
4	DATA 3	PM10 Low byte
5	DATA 4	PM10 High byte
6	DATA 5	ID byte 1
7	DATA 6	ID byte 2
8	Check-sum	Check-sum
9	Message tail	AB

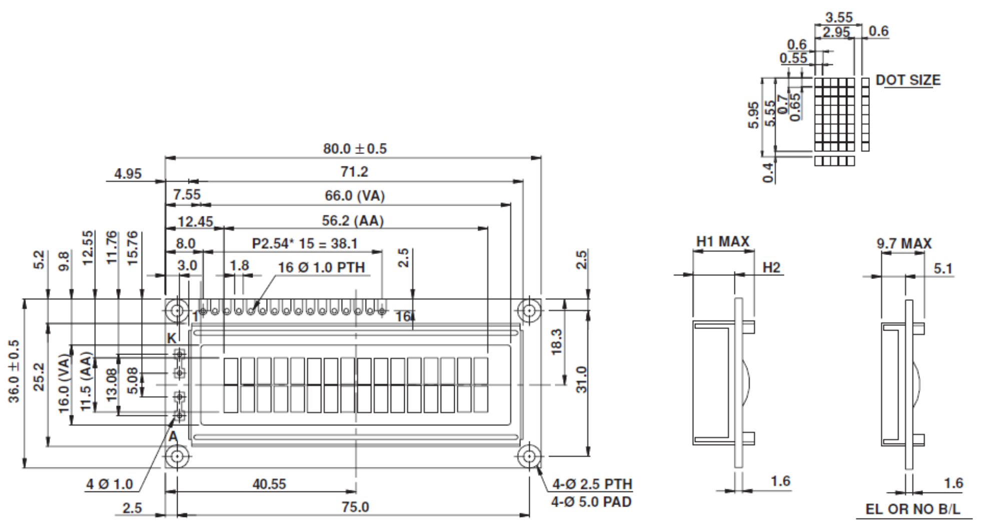

70.0000
5.0000
60.0000
SDS011
ø3.5000
11.0000
ø3.2000
ø3.2000
70.0000
55.0000
ø3.2000
TXD RXD GND 25um 5V 1um CTL
41.0000
7
1


3.5000
21.0000
1.5000

Appendix C: LCD set up and specifications

PIN NUMBER	SYMBOL	FUNCTION
1	Vss	GND
2	Vdd	+ 3V or + 5V
3	Vo	Contrast Adjustment
4	RS	H/L Register Select Signal
5	$R/\overline{W}$	H/L Read/Write Signal
6	E	H →L Enable Signal
7	DB0	H/L Data Bus Line
8	DB1	H/L Data Bus Line
9	DB2	H/L Data Bus Line
10	DB3	H/L Data Bus Line
11	DB4	H/L Data Bus Line
12	DB5	H/L Data Bus Line
13	DB6	H/L Data Bus Line
14	DB7	H/L Data Bus Line
15	A/Vee	+ 4.2V for LED/Negative Voltage Output
16	K	Power Supply for B/L (OV)

DIMENSIONS in millimeters



LED - H/L B/L

	HIGH	LOW
H1	13.2	12.1
H2	8.6	7.5

Appendix D: MATLAB code for plotting sensor data

```
%%Before running, please import data from excel sheet

Time = (Timemsfromstart - Timemsfromstart(1))/1000;
% start time from 0 seconds

%%
figure
subplot(2,1,1)
%subplot of PM25 filtered and unfiltered data
plot(Time,PM25Unfiltered,'r',Time,PM25filtered,'b');
ylim([0 999]); %%Change second value to control size of y axis
xlabel('Time (s)');
ylabel('Concentration of particles (ug/m3)');
legend('PM25 Unfiltered', 'PM25 Filtered')

subplot(2,1,2)
%subplot of PM10 filtered and unfiltered data
plot(Time,PM10Unfiltered,'r',Time,PM10filtered,'b');
ylim([0 999]);  %%Change second value to control size of y axis
xlabel('Time (s)');
ylabel('Concentration of particles (ug/m3)');
legend('PM10 Unfiltered', 'PM10 Filtered')
%%
```