```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 20, 4); // I2C address 0x27, 20 , 4
//#define C_Variable TR_C_Filtered
float C_Variable;
float propBand;
float integralTime;
float derivativeTime;

bool LM35_1=true;
bool LM35_2;
bool Forward=false;
bool Reverse=true;
bool controlAction;

String command;
int tempLocal;// read from analog pin A0
int tempRemote;// read from analog pin A3
int potA1;// pot setting  from analog pin A1
int potA3;//pot setting  from analog pin A3
float TR_C;// Remote temperature in deg C, normalizedfor PID Controller
float TL_C;//Local temperature in deg C
float TR_C_Filtered;
float TL_C_Filtered;
bool Auto=true;
bool Manual=false;
void LM35_A2();//Read Heater Temp LM35 on analog input A2, display on LCD
void LM35_A7();//Read Air Temp LM35 on analog input A7 and display to LCD
void Potentiometer(bool action);//potentiometer analog input A3,0 to 100%
void printText();// Print static text to display
void SerialPlotter();//Display variables on Arduino IDE serial plotter
float PID_output(float process, float setpoint, float Prop, float Integ,
float deriv, int Interval, bool action); // PID Controller
float SetpointGenerator(); // generate setpoint for PID
float TR_C_filterFunction(float timeConstant, float processGain,float
blockIn, float intervalTime);// filtered Heater temp sensor reading
float TL_C_filterFunction(float timeConstant, float processGain,float
blockIn, float intervalTime);//filtered Air Temp sensor reading
float DerivativefilterFunction(float timeConstant, float
processGain,float blockIn, float intervalTime);// filtered derivative
```

```
void setup()
{
    lcd.init(); //initialize the lcd
    lcd.backlight(); //open the backlight

    //********** Set the PWM pins output.**********************

    pinMode(10, OUTPUT);//switchable to either heater or fan
    pinMode(11, OUTPUT); );//switchable to either heater or fan
    // reverse acting 10 connected to heater, 11 to fan
    // forward acting 10 connected to fan, 11 to heater

    //***********Serial Monitor************************
    Serial.begin(9600);
    printText();//print static text on LCD display

    C_Variable=TR_C_Filtered;//default controlled variable is heater temp

    // ********default PID Settings********
    propBand = 7.7;
    integralTime=40;
    derivativeTime=10;

}
```

```
//*************Looping Code*********************************
void loop()
{
    float heaterSetting;// Normalized PI Controller Set Point
    float contOutNorm;//Normalized PI Controller Output
    // ***code for commands set from Arduino serial plotter***
    if (Serial.available())
    {
      command = Serial.readStringUntil('\n');
      command.trim();
      if (command.equals("Auto"))
      {
        Auto=true;
        Manual=false;
      }
      else if (command.equals("Manual"))
      {
        Manual =true;
        Auto=false;
      }
      else if (command.equals("LM35_1"))
      {
        LM35_1=true;
        LM35_2=false;
        propBand = 7.7;
        integralTime=40;
        derivativeTime=10;
      }
      else if (command.equals("LM35_2"))
      {
        LM35_2=true;
        LM35_1=false;
        propBand = 3.7;
        integralTime=120;
        derivativeTime=30;
      }
      else if (command.equals("Forward"))
      {
        Forward=true;
        Reverse=false;
      }
      else if (command.equals("Reverse"))
      {
        Forward=false;
        Reverse=true;
      }
      else
      {
      Serial.println("bad command");
      }
    Serial.print("Command: ");
    Serial.println(command);
   }
```

```
//**** End of code for Arduino serial plotter commands*****
LM35_A7();// read LM35 connected to A7

LM35_A2();// read LM35 connected to A2

Potentiometer();// manually set motor speed for

//********PI Controller Code*****************************
if(Reverse==true)
{
  controlAction =true; // PID control action to reverse acting
}
else if(Forward==true)
{
  controlAction=false;//PID control action to forward (direct) acting
}
if(LM35_1 ==true)
{
 C_Variable=TR_C_Filtered; //PID controlled variable, heater temp
}
else if (LM35_2==true)
{
 C_Variable =TL_C_Filtered; //PID controlled variable, air temp
}
```

```
    if(Auto==true )
    {
      //set the PID controller setpoint temperature range
      heaterSetting=SetpointGenerator();

      //execute the PID control, output is normalized 0 to 1.0
      contOutNorm=PID_output(C_Variable/500, heaterSetting/500,propBand,
      integralTime, derivativeTime, 2000, controlAction);

      if (controlAction==true)
      {
      // reverse acting output to pin 10 heater
      analogWrite(10,255*contOutNorm);// denormalize controller output
      }
      // forward(direct) acting output tp pin 11
      else if(controlAction==false)
      {
      analogWrite(11,255*contOutNorm);// denormalize controller output
      }

      lcd.setCursor(0, 3);// to LCD display
      lcd.print("Set Pt C ");
    }

if (Manual==true )
    {
     potA1=analogRead(A1);//read potentiometer connected to A1
     if (controlAction==true)
      {
      analogWrite(10,((float)potA1/1023*255*2.92));//sets heater output
      } else if (controlAction==false)
      {
      analogWrite(11,((float)potA1/1023*255*2.92));//sets fan output
      }

      lcd.setCursor(9 , 3);
      lcd.print("    ");
      lcd.setCursor(9 , 3);
      lcd.print (int((float)potA1/1023*100*2.92));
      lcd.setCursor(0, 3);
      lcd.print("Manual % ");
    }
    //***********End of PID Controller Code*********************

    delay(2000);// sets interval delay for PID and filters

    SerialPlotter();//plot values on Serial plotter
}
//***********End of Looping*****************************
```

```
void printText() // prints static text to display
  {

  lcd.setCursor(0, 0);
  lcd.print("Temp R C ");
  lcd.setCursor(0, 1);
  lcd.print("Temp L C ");
  lcd.setCursor(0, 2);
  lcd.print("C Out  % ");
  lcd.setCursor(0, 3);
  lcd.print("Set Pt C  ");
  lcd.setCursor(13, 0);
  lcd.print("P ");
  lcd.setCursor(13, 1);
  lcd.print("I ");
  lcd.setCursor(13, 2);
  lcd.print("D ");
  lcd.setCursor(13, 3);
  lcd.print("Fn% ");
  }

void SerialPlotter() // plots variables on Arduino IDE serial plotter
{
   if (Auto==true & Manual==false)
      {
        Serial.print(150.0);//plot horizontal line at 150
        Serial.print(",");
        Serial.print(0.0);//plot horizontal line at 0
        Serial.print(",");
        Serial.print(((float)potA1*0.4887585));//plot value of potA1,
        Serial.print(",");
        Serial.print(TR_C_Filtered);// plot heater temperature deg C
        Serial.print(",");
        Serial.println(TL_C_Filtered);// Plot air temperature deg C
      }

   else if (Manual==true & Auto==false)
      {
        Serial.print(150.0);// plot horizontal line at 150
        Serial.print(",");
        Serial.print(0.0);// plot horizontal line at 0
        Serial.print(",");
        Serial.print(int((float)potA1*0.09775*2.92));//plot potA1 value
        Serial.print(",");
        Serial.println(TR_C_Filtered);// plot heater temp
        // note not plotting air temperature in manual - very noisy
      }
}
```

```
void LM35_A2()
{
  tempRemote=analogRead(A2);//Raw heater temp in counts
  TR_C=(float)tempRemote*0.4887585;// heater temp in deg C
  TR_C_Filtered=TR_C_filterFunction(5, 1.0,TR_C, 2000); filtered temp

  //Display on LCD
  lcd.setCursor(9, 0);
  lcd.print("    ");
  lcd.setCursor(9, 0);
  lcd.print((int)TR_C_Filtered);
}

void LM35_A7()
{
  tempLocal=analogRead(A7);//Raw air temp in counts
  TL_C= (float)tempLocal*0.4887585;//air temp in deg C
  TL_C_Filtered=TL_C_filterFunction(25, 1.0,TL_C, 2000 );filtered temp

  //Display on LCD
  lcd.setCursor(9, 1);
  lcd.print("    ");
  lcd.setCursor(9, 1);
  lcd.print((int)TL_C_Filtered);

}
void Potentiometer(bool action)// fan or heater setting 0 to 100%
{
  float speedPercent;
  potA3=analogRead(A3);
  speedPercent=(float)potA3*100/1023;

  if (action==false)// forward(direct) acting
  {
   analogWrite(10,speedPercent/100*255); // output to heater
  }
  else if (action==true)// reverse acting
  {
   analogWrite(11,speedPercent/100*255); //output to fan
  }
  //Display on LCD
  lcd.setCursor(17, 3);
  lcd.print("   ");
  lcd.setCursor(17, 3);
  lcd.print(speedPercent);

}
```

```
float SetpointGenerator()// Set Point 0 to 500 deg C
{
  float setTemp;

  potA1=analogRead(A1);// read set point pot

  // while setpoint range is 0 to 500, heater can generate heat to about
  // 140 deg C. Range is set to 500 due to calibration requirement of
  // LM35s
  setTemp=(float)potA1*0.4887585;

  // display to LCD
  lcd.setCursor( 9, 3);
  lcd.print("    ");
  lcd.setCursor(9 , 3);
  lcd.print((int)setTemp);// display only integers on LCD
  return setTemp;

}
```

```
//*********Start of PID Controller algorithm*****************
float PID_output(float process, float setpoint, float Prop, float Integ,
float deriv, int Interval, bool action)
{
float Er;
static float Olderror, Cont;
static int Limiter_Switch;
static float Integral;
float derivative;
float proportional;
float deltaT;
float filteredDerivative;// limit derivative responding to noise
deltaT=float(Interval)/1000;

Limiter_Switch = 1;// prevents integral windup when output reaches
//0.0 or 1.0 (normalized)
//delay(Interval);  // Interval in msec is delta t in the integral
//and derivative  calculations
if (action==false)
  {
  Er = (process-setpoint);// forward or direct acting
  } else if (action==true)
  {
  Er=(setpoint-process); //reverse acting
  }
//Limiter switch turns integration OFF if controller is already at 100%
output or 0% output
//Prevents integral windup, where controller keeps integrating when
controller output can no longer
//affect the process.

// 2 is the interval time in seconds
if ((Cont >= 1 && Er > 0) || (Cont <= 0 && Er < 0) || (Integ >= 3600))
        Limiter_Switch = 0;
else
        Limiter_Switch = 1;

// Integ of 3600 secs esentially turns integration off
// Integral calculator
Integral = Integral + 100 / Prop / Integ * Er *deltaT * Limiter_Switch;

// Derivative calculator
derivative = 100 / Prop * deriv * (Er - Olderror) / deltaT;

// derivative filtering
filteredDerivative=DerivativefilterFunction(5, 1.0,derivative, 1000);

//proportional calculator
proportional = 100 / Prop * Er; // gain times error

Cont = proportional + Integral + filteredDerivative;
Olderror = Er;// retains previous error for deriative calculator
if (Cont > 1) // limit controller output between 0.0 and 1.0
//a normalized value
```

```
    Cont = 1;
if (Cont < 0)
    Cont = 0;

//display controller components to LCD display
lcd.setCursor(9 , 2);
lcd.print("    ");
lcd.setCursor(9 , 2);
lcd.print((int)(Cont*100.0));
lcd.setCursor(15 , 0);
lcd.print("     ");
lcd.setCursor(15 , 0);
lcd.print((int)(proportional*100.0));
lcd.setCursor(15 , 1);
lcd.print("    ");
lcd.setCursor(15 , 1);
lcd.print((int)(Integral*100.0));
lcd.setCursor(15 , 2);
lcd.print("     ");
lcd.setCursor(15 , 2);
lcd.print((int)(filteredDerivative*100.0));
return  Cont;
}
//**********End of PID *************************
```

```
//********** Exponential Filter Functions************
//time constant sets the first order delay
float TR_C_filterFunction(float timeConstant, float processGain,float
blockIn, float intervalTime)
{
float static blockOut;
blockOut=blockOut+(intervalTime/1000/(timeConstant+intervalTime/1000))*(p
rocessGain*blockIn-blockOut);
return blockOut;
}
float TL_C_filterFunction(float timeConstant, float processGain,float
blockIn, float intervalTime)
{
float static blockOut;
blockOut=blockOut+(intervalTime/1000/(timeConstant+intervalTime/1000))*(p
rocessGain*blockIn-blockOut);
return blockOut;
}
float DerivativefilterFunction(float timeConstant, float
processGain,float blockIn, float intervalTime)
{
float static blockOut;
blockOut=blockOut+(intervalTime/1000/(timeConstant+intervalTime/1000))*(p
rocessGain*blockIn-blockOut);
return blockOut;
}
```