```
// Last Modified February 15,2025
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <IRremote.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=false;
bool controlAction=true;
bool component=false;
bool tuning=true;

int tempLocal;// read from analog pin A3
int tempRemote;// read from analog pin A2
int potA0;// pot setting  from analog pin A0
int potA1;//pot setting  from analog pin A1

int receiver = 7; // Nano Signal Pin of IR receiver

float TR_C;// Remote temperature in deg C, must be normalized in argument for PI Controller
float TL_C;//Local temperature in deg C
float TR_C_Filtered;
float TL_C_Filtered;

float setTemp;// set Point 0 to 140
float speedPercent;// Fan speed 0 to 100%

bool Auto=true;
bool Manual=false;
//*********ZNOL*******************
static long unsigned count;
int m_Size_Percent= 50;
bool enablePeakDetect;
bool readOnce;
bool stopCalculate;
bool autoTune=false;
bool enableCount=false;
```

```
float yA;
float yA_old;
float slopeA;
float YA_inflect;
float startingValue;
float ZProp, ZInteg, ZDeriv;
float slopeA_Filtered;
float stepSize;
float  sampleTime=1000;
static float peakValue=0;
float slopeA_filterFunction(float timeConstant, float processGain,float blockIn, float intervalTime);
void findPeak(float currentValue);
void findPID(float peakSlope, int timeT2, float yA_Peak, float yA_Start, float M_step, float *PB, float *I,
float *D);

//**********************************
void LM35_A2();//Read the Remote LM35 on analog input A1 and display on LCD
void LM35_A3();//Read the Local LM35 on analog input A0 and display to LCD
void Potentiometer(bool action);// Read the potentiometer setting as an analog input and display as 0 to
100%
void printText();// Print static text to LCD display
void SerialPlotter();

void translateIR(); // takes action based on IR code received

float PID_output(float process, float setpoint, float Prop, float Integ, float deriv, int Interval, bool action);
float SetpointGenerator();
float TR_C_filterFunction(float timeConstant, float processGain,float blockIn, float intervalTime);
float TL_C_filterFunction(float timeConstant, float processGain,float blockIn, float intervalTime);
float DerivativefilterFunction(float timeConstant, float processGain,float blockIn, float intervalTime);

IRrecv irrecv(receiver);    // create instance of 'irrecv'
decode_results results;     // create instance of 'decode_results'

void setup()
{
  lcd.init(); //initialize the lcd
  lcd.backlight(); //open the backlight
  irrecv.enableIRIn(); // Start the receiver
  //********** Set the PWM pins output.**********************
  pinMode(10, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, INPUT);
```

```
//***********Serial Monitor*******************************
   Serial.begin(9600);
   Serial.println("IR Receiver Button Decode");
   irrecv.enableIRIn(); // Start the receiver

   //*******Print LCD Static Text*****************************
    printText();
   //*****************************************************
   C_Variable=TR_C_Filtered;
   propBand =9;
   lcd.setCursor(15 , 0);
   lcd.print((int)(propBand));
   integralTime=28;
   lcd.setCursor(15 , 1);
   lcd.print((int)(integralTime));
   derivativeTime=14;
   lcd.setCursor(15 , 2);
   lcd.print((int)(derivativeTime));
 }
void loop()
{
   float heaterSetting;// Normalized PID Controller Set Point
   float contOutNorm;//Normalized PID Controller Output
//************Infra Red Remote Control******************
if (irrecv.decode() )// have we received an IR signal?
    {
    irrecv.decodedIRData.decodedRawData ;
    translateIR();
    irrecv.resume(); // receive the next value
    Serial.print("results = ");
    Serial.println(irrecv.decodedIRData.decodedRawData, HEX);
    }

   LM35_A3();// read LM35 connected to A3
   LM35_A2();// read LM35 connected to A2
   //********Manual Fan Speed Setting Code **********************
   Potentiometer(controlAction);// manually set motor speed for OCR1B
   ZNOL();
   //*********End of Manual Speed Setting Code*****************
```

```
//********PID Controller Code*****************************
    if(LM35_1 ==true)
   {
    C_Variable=TR_C_Filtered;
   }
   else if (LM35_2==true)
   {
    C_Variable =TL_C_Filtered;
   }
   if(Auto==true )
   {
     heaterSetting=SetpointGenerator();// this establishes the PID Setpoint ... range is 0 to 110 deg C
     contOutNorm=PID_output(C_Variable/500, heaterSetting/500,propBand, integralTime,
derivativeTime, 1000, controlAction); //normalized 0 to 1.0 of PID controller output ... PB=800, Tint = 32,
Td=0, interval =100msec
     if (controlAction==true)
     {
     analogWrite(10,255*contOutNorm);// converts PID controller output toa PWM value
      }
     else if(controlAction==false)
     {
     analogWrite(6,255*contOutNorm);// converts PID controller output toa PWM value
     analogWrite(10,(int)(speedPercent/100*255)); //write to heater(direct acting), acts as heat load
     }
     lcd.setCursor(0, 3);
     lcd.print("Set Pt C ");
   }
   if (Manual==true )
    {
    potA0=analogRead(A0);

    if (controlAction==true & autoTune==false)
      {
      analogWrite(10,((float)potA0/1023*255*3.59));// 3.59 added to compensate for 1.39 volts at top of
pot
      }
      else if (controlAction==false & autoTune==false)
      {
      analogWrite(6,((float)potA0/1023*255*3.59));
      analogWrite(10,(int)(speedPercent/100*255));
      }
   lcd.setCursor(9 , 3);
   lcd.print("   ");
   lcd.setCursor(9 , 3);
```

```
   lcd.print (int((float)potA0/1023*100*3.59));// Manual Value
   lcd.setCursor(0, 3);
   lcd.print("Manual % ");
   }
   //***********End of PID Controller Code*********************
   yA_old=yA;
   SerialPlotter();//plot values on Serial plotter
   delay(1000);
 }

void printText()
 {
 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(12, 2);
 lcd.print(" D ");
 lcd.setCursor(12, 3);
 lcd.print(" Ds% ");
 }
void SerialPlotter()
{     if (Auto==false & Manual == true & autoTune==false)
      {
      Serial.print(0.0);//plot value of potA0, either Set Point or Manual
      Serial.print(",");
      Serial.print(100.0);//plot value of potA0, either Set Point or Manual
      Serial.print(",");
      Serial.print((float)potA0/1023*100*3.59);//plot value of potA0,  or Manual
      Serial.print(",");
      Serial.print(TL_C_Filtered);// Plot remote LM35 which is controlled variable 0 to 500
      Serial.print(",");
      Serial.print(TR_C_Filtered);// Plot remote LM35 which is controlled variable 0 to 500
      Serial.print(",");
      Serial.print(stepSize);
```

```
        Serial.print(" = step ");
        Serial.println(",");
        }
        else if (Auto==true & Manual==false& autoTune==false)
        {
        Serial.print(0.0);//plot value of potA0, either Set Point or Manual
        Serial.print(",");
        Serial.print(100.0);//plot value of potA0, either Set Point or Manual
        Serial.print(",");
        Serial.print(setTemp);//plot value of potA0,  or Manual
        Serial.print(",");
        Serial.print(TL_C_Filtered);// Plot remote LM35 which is controlled variable 0 to 500
        Serial.print(",");
        Serial.print(TR_C_Filtered);// Plot remote LM35 which is controlled variable 0 to 500
        Serial.print(",");
        Serial.print(stepSize);
        Serial.print(" = step ");
        Serial.println(",");
        }
        else if (Auto==false & Manual == true & autoTune==true)
        {
        Serial.print(0.0);//plot value of potA0, either Set Point or Manual
        Serial.print(",");
        Serial.print(100.0);//plot value of potA0, either Set Point or Manual
        Serial.print(",");
        Serial.print((float)potA0/1023*100*3.59);//plot value of potA0,  or Manual
        Serial.print(",");
        Serial.print(slopeA_Filtered);
        Serial.print(",");
        Serial.print(TR_C_Filtered);// Plot remote LM35 which is controlled variable 0 to 500
        Serial.print(",");
        Serial.print(stepSize);
        Serial.print(" = step ");
        Serial.println(",");
        }
}
```

```
void LM35_A2()
{
 tempRemote=analogRead(A2);
 TR_C=(float)tempRemote*0.4887585;
 TR_C_Filtered=TR_C_filterFunction(5, 1.0,TR_C, 1000);//filter time constant is first argument in seconds
 if(autoTune==false)
 {
 lcd.setCursor(9, 0);
 lcd.print("    ");
 lcd.setCursor(9, 0);
 lcd.print((int)TR_C_Filtered);
 }
}
void LM35_A3()
{
 tempLocal=analogRead(A3);
 TL_C= (float)tempLocal*0.4887585;
 TL_C_Filtered=TL_C_filterFunction(5, 1.0,TL_C, 1000 );
 if(autoTune==false)
 {
 lcd.setCursor(9, 1);
 lcd.print("    ");
 lcd.setCursor(9, 1);
 lcd.print((int)TL_C_Filtered);
 }
}

void Potentiometer(bool action)// fan setting 0 to 100%
{
 potA1=analogRead(A1);
 speedPercent=(float)potA1*100/1023;
 if (action==false)//direct acting
 {
  analogWrite(10,(int)(speedPercent/100*255)); //write to heater(direct acting), acts as heat load
 }
 else if (action==true)//reverse acting
 {
  analogWrite(6,(int)(speedPercent/100*255));// write to fan (reverse acting)
 }
  lcd.setCursor(17, 3);
 lcd.print("   ");
 lcd.setCursor(17, 3);
 lcd.print ((int) (speedPercent));
}
```

```
float SetpointGenerator()// Set Point 0 to 140 dg C
{
  potA0=analogRead(A0);
  if (potA0<=300)
  {
    setTemp=(float)potA0*0.4917;// generates a temp setpoint of 0 to 140 deg C  , pot voltage 0 to 1.4V
and count of 0 to 284 counts
    if(autoTune==false)
      {
      lcd.setCursor( 9, 3);
      lcd.print("    ");
      lcd.setCursor(9 , 3);
      lcd.print((int)setTemp);
      }
    return setTemp;
  } else
  {
    setTemp=146;
    if(autoTune==false)
    {
      lcd.setCursor( 9, 3);
      lcd.print("    ");
      lcd.setCursor(9 , 3);
      lcd.print((int)setTemp);
      return setTemp;
    }
  }
}

float PID_output(float process, float setpoint, float Prop, float Integ, float deriv, int Interval, bool action)
{
float Er;
static float Olderror, Cont, old_Process;
static int Limiter_Switch;
static float Integral;
float derivative;
float proportional;
float deltaT;
float filteredDerivative;
deltaT=float(Interval)/1000;
Limiter_Switch = 1;
```

```
if (action==false)
 {
 Er = (process-setpoint);// 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.
// 1 is the interval time in seconds

if ((Cont >= 1 && Er > 0) || (Cont <= 0 && Er < 0) || (Integ >= 3600))
      Limiter_Switch = 0;
else
      Limiter_Switch = 1;

Integral = Integral + 100 / Prop / Integ * Er *deltaT * Limiter_Switch;// Integral calculator
derivative = 100 / Prop * deriv * (old_Process-process) / deltaT;// Derivative calculator, derivative on
process, not error to eliminate derivative action on setpoint
filteredDerivative=DerivativefilterFunction(2, 1,derivative, 1000);
proportional = 100 / Prop * Er;// Proportional calculator

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

if (Cont < 0)
   Cont = 0;
if(autoTune==false)
 {
 lcd.setCursor(9 , 2);
 lcd.print("    ");
 lcd.setCursor(9 , 2);
 lcd.print((int)(Cont*100.0));
 }
if(component==true & autoTune==false)
 {
   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;
}

//*****************Filtering*************************************
float TR_C_filterFunction(float timeConstant, float processGain,float blockIn, float intervalTime)
{
float static blockOut;
blockOut=blockOut+(intervalTime/1000/(timeConstant+intervalTime/1000))*(processGain*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))*(processGain*blockIn-
blockOut);
return blockOut;
}

float DerivativefilterFunction(float timeConstant, float processGain,float blockIn, float intervalTime)
{
float static blockOut;
blockOut=blockOut+(intervalTime/1000/(timeConstant+intervalTime/1000))*(processGain*blockIn-
blockOut);
return blockOut;
}
//*************ZNOL Functions*******************
float slopeA_filterFunction(float timeConstant, float processGain,float blockIn, float intervalTime)
{
float static blockOut;
blockOut=blockOut+(intervalTime/1000/(timeConstant+intervalTime/1000))*(processGain*blockIn-
blockOut);
return blockOut;
}
```

```
void findPeak(float currentValue)
{
 if (readOnce==true)
 {
  startingValue=yA;
 }

 if (currentValue>peakValue)
 {
  peakValue=currentValue;
  YA_inflect=yA;
  readOnce=false;
  findPID(slopeA, count, YA_inflect, startingValue, m_Size_Percent, &ZProp, &ZInteg, &ZDeriv);
 }
 if (autoTune==true)
 {
 lcd.setCursor(12, 0);
 lcd.print("M = ");
 lcd.setCursor(17 , 0);
 lcd.print("   ");
 lcd.setCursor(17 , 0);
 lcd.print((int)m_Size_Percent);
 lcd.setCursor(12, 2);
 lcd.print("Sl = ");
 lcd.setCursor(17 , 2);
 lcd.print("   ");
 lcd.setCursor(17 , 2);
 lcd.print((int)slopeA_Filtered);
 lcd.setCursor(12 , 3);
 lcd.print("time=");
 lcd.setCursor(17 , 3);
 lcd.print("   ");
 lcd.setCursor(17 , 3);
 lcd.print(count);
 }

}
```

```
void findPID(float peakSlope, int timeT2, float yA_Peak, float yA_Start, float M_step, float *PB, float *I,
float *D)
{
 float R, L, timeT1;// timeT1 is base of triangle formed by peakSlope and (yA_Peak-yA_Start)
 R=(0.24*peakSlope); //reaction rate in %/min, (6*peakSlope/5)/500*100
 timeT1=(yA_Peak-yA_Start)/(peakSlope/50.0);
 L=timeT2-timeT1;//timeT2 is start of step to point of inflectio, timeT1 is time from intersection of
tangent to point of inflection
  *PB=1.38833*L*R/M_step;//83.3*(L/60)*R/M_step
 if(*PB<1) // don't show 0 when PB < 1%
   *PB=1;
 *I=2.0*L;
 *D=0.5*L;
  lcd.setCursor(12 , 1);
  lcd.print("L = ");
  lcd.setCursor(17 , 1);
  lcd.print("  ");
  lcd.setCursor(17 , 1);
  lcd.print((int)L);
   lcd.setCursor(0, 0);
   lcd.print("PB = ");
   lcd.setCursor(6, 0);
   lcd.print((int)ZProp);
   lcd.setCursor(0, 1);
   lcd.print("Int = ");
   lcd.setCursor(6 , 1);
   lcd.print((int)ZInteg);
   lcd.setCursor(0 , 2);
   lcd.print("Der = ");
   lcd.setCursor(6 , 2);
   lcd.print("    ");
   lcd.setCursor(6 , 2);
   lcd.print((int)ZDeriv);
}
```

```
void ZNOL()
{
if(enableCount==true)
 {
 count++; //start counting seconds based on interval(1000)
 }
  //************************************
  if(LM35_1 ==true)
  {
   yA=TR_C_Filtered;
  }
  else if (LM35_2==true)
  {
   yA=TL_C_Filtered;
  }
  slopeA=50*(yA-yA_old)/(sampleTime/1000);
  slopeA_Filtered=slopeA_filterFunction(10, 1,slopeA, 1000);  // was set to 5 , changed
  if(slopeA_Filtered>50)//prevents huge initial filtered slope
     slopeA_Filtered=50;
  if(slopeA>50)//prevents huge initial slope
     slopeA=50;
   if(enablePeakDetect==true)
   {
   findPeak(slopeA_Filtered);//for stepresponse, find the peak value of slope which yields inflection
point
   }
 if ( autoTune==false )
   {

   potA0=analogRead(A0);
   analogWrite(10,((float)potA0/1023*255));// remove jumper to apply full 5 volts
   //stepSize=((float)potA0/1023*100;// for purpose of plotting
   }
  if (autoTune==true )// allows a step from different settings of the pot.
   {
   potA0=analogRead(A0);
   //analogWrite(10,((float)potA0/1023*255+150));// add 150 counts  creates a step of 25%
(64/255*100)
   stepSize=((float)potA0/1023*255+(float)m_Size_Percent/100*255)/255*100;// for purpose of
plotting
   }
}
```

```
//********************IR Control*****************************************
void translateIR() // takes action based on IR code received
{
 switch(irrecv.decodedIRData.decodedRawData)

 {
 case 0xBA45FF00: //POWER
 lcd.setCursor(15 , 0);
 lcd.print("    ");
 lcd.setCursor(15 , 0);
 lcd.print((int)(propBand));
 lcd.setCursor(15 , 1);
 lcd.print("    ");
 lcd.setCursor(15 , 1);
 lcd.print((int)(integralTime));
 lcd.setCursor(15 , 2);
 lcd.print("    ");
 lcd.setCursor(15 , 2);
 lcd.print((int)(derivativeTime));
 printText();
 component=false;
 tuning=true;
 break;

 case 0xB847FF00: //FUNC/STOP
 lcd.setCursor(15 , 0);
 lcd.print("    ");
 lcd.setCursor(15 , 1);
 lcd.print("    ");
  lcd.setCursor(15 , 2);
 lcd.print("    ");
 printText();
 component=true;
 tuning=false;
 break;

 case 0xB946FF00: //VOL+ Direct Acting
 controlAction=false;
 break;

 case 0xBB44FF00: //FAST BACK - Auto
 Auto=true;
 Manual=false;
 break;
```

```
case 0xBF40FF00: //Pause
propBand = ZProp;
integralTime=ZInteg;
derivativeTime=ZDeriv;
lcd.setCursor(15 , 0);
lcd.print("     ");
lcd.setCursor(15 , 0);
lcd.print((int)propBand);
lcd.setCursor(15 , 1);
lcd.print("     ");
lcd.setCursor(15 , 1);
lcd.print((int)(integralTime));
lcd.setCursor(15 , 2);
lcd.print("     ");
lcd.setCursor(15 , 2);
lcd.print((int)(derivativeTime));
printText();
break;

case 0xBC43FF00: //FAST FORWARD - Manual
Auto=false;
Manual=true;
break;

case 0xF807FF00: //DOWN - T Remote
LM35_1=true;
LM35_2=false;
break;

case 0xEA15FF00: //VOL-Reverse
controlAction=true;
break;

case 0xF609FF00: //UP - T Local
LM35_2=true;
LM35_1=false;
break;

case 0xE619FF00: //EQ remove step
autoTune=false;
enablePeakDetect=false;
count=0;
stepSize=((float)potA0/1023*255)/255*100;
```

```
printText();
enableCount=false;
break;

case 0xF20DFF00: //ST/REPT
printText();
break;

case 0xE916FF00: //0 start Autotune
lcd.setCursor(0, 0);
lcd.print("                ");
lcd.setCursor(0, 1);
lcd.print("                ");
lcd.setCursor(0, 2);
lcd.print("                ");
lcd.setCursor(0, 3);
lcd.print("                ");
autoTune=true;
enablePeakDetect=true;
readOnce=true;
count=0;
enableCount=true;
break;

case 0xF30CFF00: //1- Prop Band - 2
if(tuning==true)
{
propBand=propBand-1;
if(propBand< 1)
  propBand=0.1; // for On/Off Control
lcd.setCursor(15 , 0);
lcd.print("    ");
lcd.setCursor(15 , 0);
lcd.print((int)(propBand));
}
break;

//case 0xE718FF00: //2
//break;

case 0xA15EFF00: //3 -Prop Band*2
if(tuning==true)
{
propBand=propBand+1;
```

```
lcd.setCursor(15 , 0);
lcd.print("     ");
lcd.setCursor(15 , 0);
lcd.print((int)(propBand));
}
break;

case 0xF708FF00: //4 Integ_Time-2
if(tuning==true)
{
integralTime=integralTime-2;
lcd.setCursor(15 , 1);
lcd.print("    ");
lcd.setCursor(15 , 1);
lcd.print((int)(integralTime));
}
break;

case 0xE31CFF00: //5
m_Size_Percent=m_Size_Percent+5;
break;

case 0xA55AFF00: //6 -Integ Time+2
if(tuning==true)
{
integralTime=integralTime+2;
lcd.setCursor(15 , 1);
lcd.print("    ");
lcd.setCursor(15 , 1);
lcd.print((int)(integralTime));
}
break;

case 0xBD42FF00: //7 - Deriv Time
if(tuning==true)
{
derivativeTime=derivativeTime-1;
if(derivativeTime<0)
  derivativeTime=0;
lcd.setCursor(15 , 2);
lcd.print("     ");
lcd.setCursor(15 , 2);
lcd.print((int)(derivativeTime));
}
```

```
 break;

case 0xAD52FF00: //8
m_Size_Percent=m_Size_Percent-5;
break;

 case 0xB54AFF00: //9 - DerivTime+1
 if(tuning==true)
 {
 derivativeTime=derivativeTime+1;
 lcd.setCursor(15 , 2);
 lcd.print("     ");
 lcd.setCursor(15 , 2);
 lcd.print((int)(derivativeTime));
 }
 break;

 //case 0xFFFFFFFF: //REPEAT
 //break;

 default:
 Serial.println(" other button   ");
 }// End Case
}
```